Connection-Oriented Architecture Framework for Enterprise-class Web Services on Multi-Core Servers

THESIS

Submitted in partial fulfilment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY
by

P. V. SURESH

Under the Supervision of

Dr. G. Karthikeyan

2005PHXF025P

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)

2014

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)

CERTIFICATE

This is to certify that the thesis entitled Connection-Oriented Architecture Framework for Enterprise-class Web Services on Multi-Core Servers ,
which is submitted for award of Ph.D. Degree of the Institute, embodies original work done by him under my supervision.

Signature in full of the Supervisor: G Karthikeyan
Name in capital block letters: Dr. G. KARTHIKEYAN
Designation: Senior Manager - Projects

Date:

ABSTRACT

This dissertation proposes an architectural framework for efficiently utilizing the abundant computing power provided by Multi-core processors to compute the workloads of web service based enterprise applications. The enterprise class applications are constructed and orchestrated using web-services through plethora of design patterns, and hosted under single or multi-threaded infrastructure containing various hardware and operating system configurations. These applications contain short lived and long-lived transactions/sessions, and thereby fulfilling various enterprise needs such as simple data entry user interfaces to highly rich and interactive drill-down dashboards. They have the stringent maintainability, reliability and availability requirements, and have extensive change management processes. Currently multiple architectural frameworks and programming models are evolving to exploit the multiple threads available in Multi-core servers. These frameworks assume the presence of parallel threads in Multi-core and suggest methods and algorithms for exploiting this parallelism. Typical suggestions of these frameworks to exploit the Multi-core are: modify the application to identify parallel operations in the business logic, recompile the application, or modify the underlying operating system implementations. These frameworks assume that the hardware infrastructure is homogenous, and hence they have limited importance to constraints that arise due to eco-system for Multi-core, such as disk memory size, network latency etc. This narrow and specialized treatment to exploit the parallel threads in Multi-core is excellent for developing green field applications, but may not be directly amenable to existing enterprise applications. The enterprises applications are hosted on heterogeneous infrastructure with various hardware and software configurations. To insert a Multi-core server into the enterprise hardware infrastructure, holistic treatment is required to address the needs of enterprise architecture. The architecture needs to be inclusive in nature to the existing enterprise requirements while marrying the new innovations of Multi-core. Additionally, the architecture needs to consider the constraints imposed by its other sub-systems like network, memory. Thus, in the context of using Multi-core systems in the enterprise, there is a need for a formal mechanism or a framework that can provide an improved performance to the existing applications without requiring any change or requires a minimal change to the existing applications or its underlying operating system.

A six module software framework is proposed, called the Connection-Oriented Architecture Framework on Multi-core servers (COAF) for web services, for designing generic enterprise applications. The web service based application developed or hosted using COAF is able to provide the best user perceived application performance while optimally leveraging the processing capabilities available in the Multi-core. If the web service application is pre-written, then COAF guides the system administrator with best infrastructure configuration, so that application is able to leverage the Multi-core capabilities without changing the application code. The six modules are namely, Resource Management module, Behaviour Information module, System Management module, Threshold Configuration module, Core Scheduler module, and Notification Framework module. These six modules of COAF together embed two key architectural concepts namely the notion of "Connection-Orientation" and the notion of "Infrastructure Awareness". The notion of Connection-Orientation is achieved by series of steps namely, placement of services on servers with specific application configurations, remembering and mapping the previous execution contexts of the service requests, and contextually dispatching the service request based on existing load on those servers. The notion of Infrastructure Awareness is achieved by observing and measuring the performance metrics of the application at the hardware and operating system level and correlating these metrics against the pre-computed threshold configuration values for that application.

Connection-Orientation is established by the scheduler, which intercepts the incoming service request, and converts the request target address to the server location where the service is deployed. For each service, scheduler creates and maintains two data structures namely, a state object called "SchedulerState" and a lookup service called "ServiceMap". SchedulerState provides the information about how busy the overall system is, and the availability of the servers for processing the request. Based on the availability and the utilization of the core / server, the scheduler decides the destination address of the server and updates that information in "SchedulerState". "ServiceMap" provides the server location where the service is deployed.

Infrastructure Awareness in Multi-core is gained through the continuous feedback received from the run-time behaviour of the system. The run time information in turn helps to model the behaviour of the system for the system metrics against the configuration settings. Over the course of time through the run time metrics, the threshold values for the system configurations

are identified. Then the system configurations are setup for optimum performance and the servers are booted up with these system configurations.

The six modules of COAF framework are implemented using Java based web services. When the web service application is executed, the behaviour of the execution is observed at operating system level and "Cache miss" is measured. Based on the Cache miss measurements, two new parameters are derived namely Core Configuration Effectiveness (CCE) and Intra Process Efficiency (IPE) that define the performance of a generic enterprise application. These two parameters CCE and IPE help us to compare, to contrast, and to tune the application for improved performance under Multi-core environment without modifying the application or the underlying operating system. We observed an association between input parameters such as number of client threads, number of cores, memory size, application characteristics etc. with the performance the underlying Multi-core system for a given application. We then calculated the threshold values for those configurations, and demonstrated the opportunities to improve utilization of available cores, and associated user-perceived improvements up to 20% in processing workloads without modifying the applications. We also demonstrated the flexibility of COAF framework to accommodate various deployment scenarios and infrastructure configurations, namely Client-Server in same network, Client-Server in different networks, in-memory access, remote-access, different architectures (32 bit and 64 bit), different memory configurations (1 GB to 8 GB memory of RAM), and different core combinations (1 to 32 cores processing capacities) under various application loads.

This holistic six module approach of designing the application architecture under COAF thus enables us to effectively model, build and deploy the enterprise web services on the Multi-core and other emerging hardware / software platforms without modifying the existing applications.

ACKNOWLEDGEMENTS

This dissertation has been achieved with the encouragement, support, and assistance I received from many remarkable people. I would like to offer my sincere gratitude to them.

First of all, I would like to thank my advisor Prof. Karthikeyan Ganesan, for his support, guidance and an exceptional research environment provided by him along the way of this endeavour. I deeply appreciate how much he contributed with his keen insight and extensive experience. His advice was always invaluable contribution to my research.

I would also like to thank my management, especially Mr. Yoosuf Mohammad, Vice President, Cognizant Technology Solutions, for generously offering time, morale support, guidance and good will throughout the preparation of this dissertation.

I want to thank Dr. Sharad Shrivatsava. Dr. Navneet Goyal, Mr. Dinesh Kumar, Ms. Monica Sharma, the staff of Research and Consultancy Division, the staff of Student Welfare Division, and members of Doctoral Advisory Committee, BITS Pilani, for guiding me and helping me throughout the research program.

I am very thankful to Mr. Sravan Desikan, Mr. Muralidharan Valuthur, and staff of Global Technology Office, Cognizant Technology Solutions, for helping me set up the test environment.

Through my company Cognizant Technology Solutions, I am very fortunate to directly work on the state of the art systems in web services based online and SaAS businesses such as eBay, ETRADE, SAP, Amazon, and Intuit, and I want to thank those outstanding architects for productive discussions on various aspects of my research.

Finally, I am especially grateful to my family and friends for their contributions throughout my graduate studies. I have received wonderful support from my parents, Chandra and Venugopal, my wife Radhika, my lovely daughters Shruthi and Meera. Their constant love and encouragement were invaluable.

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ABBREVIATIONS AND ACRONYMS

ACK	Acknowledgment
ADU	Abstract Data Unit
ANOVA	Analysis of Variance
Apache Axis2	XML based Web service framework
Apache Webserver	Apache HTTP Server
API	Application Programming Interface
B2B	Business to Business
B2C	Business to customer
BG/L	IBM Blue Gene/L
BSE	Bombay Stock Exchange
BSP	Binary Space Partitioning
CGE	Carrier Grade Edition
C/S	Client/Server model
Click	Modular Packet Router architecture model
Cluster	Cluster is the sequence of the same process executing continuously on a core
CMP	Chip Multi Processor
CORBA	Common Object Request Broker Architecture
COAF	Connection-Oriented Architecture Framework on Multi-core
COAF-MC	Refer COAF
Core configuration	Arrangement of number of cores in a single processing cluster
CPU	Central Processing Unit
CRM	Customer Relationship Management
CRUD	Create, Read, Update and Delete
DCOM	Distributed Component Object Model
DDRAM	Double Data Rate Synchronous DRAM
DIS	Distributed Interactive Simulation
DIMM	Dual In-line Memory Module - series of DRAM

dispqlen_by_cpu	DTrace probe for measuring dispatcher queue length
DNS	Domain Name Service
DOM	Document Object Model
DRAM	Dynamic Random Access Memory
Dryad	Distributed Data-Parallel Programs from Sequential Building Blocks
DSM	Distributed Shared Memory
DSS	Decision Support System
FBDIMM	Fully Buffered DIMM
FPU	Floating point units
GPU	Graphical Processing Unit - similar to a CPU with specific support for graphics
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IIOP	Internet Inter ORB Protocol
ILP	Instruction Level parallelism
IP	Internet Protocol (Layer 3)
J2EE	Java 2 Enterprise Edition
JDBC	Java Data-Base Connectivity
JMeter	A load testing tool
JMS	Java Message Service API for sending messages between two or more clients.
JVM	Java Virtual Machine
LAMP	Linux, Apache, MySQL, and PHP
LAN	Local Area Network
LDoms	Logical Domains 1.1
LTT-CONTROL	Linux Trace Toolchain Control
LTTng	Linux Trace Toolkit Next Generation
LTTV	Linux Toolkit Trace viewing
MVC	Model-View-Control
MySQL	Relational Database system
NaradaBrokering	Notification Framework module

NSE	National Stock Exchange
OLTP	Online transaction processing
OMG	Object Management Group
ORB	Object Request Broker
OS	Operating Systems
P2P	Peer-to-Peer
PHP	General purpose scripting language for web applications
Python	General purpose language for web applications
RAM	Random Access Memory
RDBMS	Relational Database Management System
Reference threshold	Initial threshold values
REST	Representational State Transfer
RPC	Remote Procedure Call
SaAS	Software as a Service
SAX	Simple API for XML
Schedule	Plan executed by Operating System to process the workload
SDRAM	Synchronous Dynamic Random Access Memory
SEDA	Staged Event Driven Architecture
SGML	Standard Generalized Markup Language
Slot	Time period in which the process is executed
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
Solaris Containers	Combination of system resource controls and the boundary separation provided by zones
Solaris Zones	Completely isolated virtual servers running one operating system instance
SOAP	Simple Object Access Protocol
StagedServer	Architecture style using Cohort scheduling method
syscall_by_proc	DTrace probe
SystemTap	Kernel instrumentation framework
TCO	Total cost of ownership
TCP	Transmission Control Protocol

Time slot	Same as slot
Throughput	Average rate of successful message delivery over a communication channel
TLP	Thread Level Parallelism
TS1	Naming convention given for time slot
UDDI	Universal Description, Discovery, and Integration
UDP	User Datagram Protocol; lost packets and out of order packets are not handled
UI	User Interface
UID	Unique Identifier
URI	String of characters used to identify a name or a resource on the Internet.
URL	Uniform Resource Locator (an URI and the mechanism for retrieving it)
WAMP	Windows Apache MySQL PHP
WS	Web Services
WSDL	Web Services Description Language
WSDM	Web Services Distributed Management
WSR	WS-Resource
WSRF	Web services Resources Framework
XML	Extensible Markup Language
XML-RPC	Remote Procedure Call protocol encoded in XML
Zones	See Solaris Zones

TABLE OF CONTENTS

1 INTRODUCTION

1.1 INTRODUCTION

The enterprise computing infrastructure has been a test-bed for a continuous architectural evolution from mainframes, to client/server, multi-tier, peer-to-peer, clusters, grids and most recently to clouds. The Internet and its explosive growth have provided a new opportunity for enterprises to deploy the applications on the web using simple access protocols like HTTP, so that customers can consume the applications very easily using the universal front end like browser. Web service standards and web service interfaces provided an opportunity for developers to construct the applications very easily and deploy these applications on the web using the universal service definitions like web services in the global scale. Among the other trends, two major trends in computing, one in software and the other in hardware that are influencing the adoption of web services.

- The rapid adoption of service-oriented architectures [198], [202], [211] based on web services to construct the software [193] define the software trend as outlined in Section 1.2.
- The evolution of Multi-core based architectures with large number of computing threads to deploy the services on a massive scale [29], [93], [158] defines the hardware trend as outlined in Section 1.3.

1.2 INFLUENCING THE ADOPTION OF WEB SERVICES – SOFTWARE PERSPECTIVE

Today, enterprises use the web in numerous ways for hosting its applications; from sharing the company's marketing brochures about its products and services to end customers; to collaborating with its suppliers in real-time for various business processes [100] such as supply chain management; and to providing access to its internal applications such as time-sheet systems to its employees. Typically, these applications contain multiple business processes and each business process is a set of logically related activities precisely choreographed and executed [83], [184], [187], [198] to achieve a well defined business outcome. Some examples to represent these business processes are - processing a credit claim, hiring a new employee, ordering goods from a supplier, creating a marketing plan, processing and paying an insurance

claim, etc. An activity is an element that performs a specific function within a business process. Activities can be simple and "short-lived" as in sending or receiving a message, or as complex and "long-lived" as in coordinated execution of multiple activities. The business process can be synchronous or asynchronous.

These enterprise applications need to be enhanced continuously for various reasons such as changing customer behaviour, and application usage patterns. For example, today the customer has access to compare the price of a product from various web sites like Amazon [7] and eBay [67]; a recruiter can verify the social graph of a candidate in facebook [75], linkedin.com [145] and twitter.com [226] before interviewing to the candidate etc. In the above scenarios, the enhancement for a customer support application is to have an access to external web sites for price comparison, and for the talent management application to have an API call to LinkedIn web site for verifying the background of the candidate. Hence, these enterprise systems needed to be constructed in such a way that these applications adapt to the business agility. Web service standards framework [54], [242], [246], [248] and Service oriented architectures [139], [228] provide the architectural framework and standards to construct and to deploy these enhancements in a rapid speed, thus enabling the business to accommodate them in an agile manner.

Service oriented architectures introduced a simple and elegant construction model for creating dynamic-distributed applications [24], [65], [100], [241]]. These architectures use a simple textual representation defined by Web service standards using technologies such as XML [31], [38] to declare a business intent in the form of a web service, and this web service could be deployed across the Internet using open network standards such as TCP/IP, HTTP [79]. Now, web services are emerging as a standard framework for constructing and assembling business processes and business workflows.

Enterprise web services are built on service-oriented principles [71], such as Formal contract, loose coupling, encapsulation, composability, reusability, autonomy and discoverability. Essentially, each web service based application becomes an accessible web service component that is described using open standards. Web services framework as shown in Figure 1-1 is divided into three major areas and addressed by three standards respectively. They are:

Communication protocol – Simple Object Access Protocol (SOAP) [245] enables communication among web services;

Service description – Web Services Description Language (WSDL) [247] provides a formal, computer-readable description of web services;

Service discovery – Universal Description, Discovery, and Integration (UDDI) [177] directory, is a registry of Web services descriptions;

In this standard driven framework, a web service is an access endpoint to data and functional resources. The web service endpoint is published in a UDDI directory, wherein a client discovers its location. The available data and operations are described in XSD (XML Schema Definition) [38] and WSDL. The client generates invocation stubs that perform run-time data conversion to SOAP message format. The client invokes the service (using a transport protocol, like HTTP) and the service is executed and the client receives the response after the completion of execution. Presently web services standards cover various aspects of enterprise applications such as data representation, transactions, security, messaging, communication between systems, Resource provisioning, resource definition and discovery, business event handling, etc. The concept of using of web services to design the enterprise applications led to powerful business models and interaction models in the computing world, instances of two such models are Business to customer (B2C), and Business to Business (B2B).

Web service based applications such as Amazon [7], eBay [67], Facebook [75], Google [98], Microsoft Live [161], MySpace [165], Twitter [226], Yahoo [253], etc. have demonstrated the power of web services to handle billions of transactions every day. These applications are constructed in connection-less architectural style, using fine grained web services that are rendered over the web. These architectures encompass extraordinary scalability and availability to handle the continuously running business transactions. This ability of web service based platforms motivates enterprises to consider web services as architectural construct for developing the applications.

Another compelling trend is the evolution of new paradigms such as utility computing, grid computing, cloud computing, platform as a service, software as service etc., which aims to shift the enterprise investments from fragmented information technology assets to a centralized utility service [42], thereby achieve the economies of scale. Table 1-1 depicts the total cost of ownership (TCO) shown by McKinsey Quarterly for a sample deployment of customer relationship management (CRM) software for 200 seat license [64], and the expenses are given in thousands of dollars.

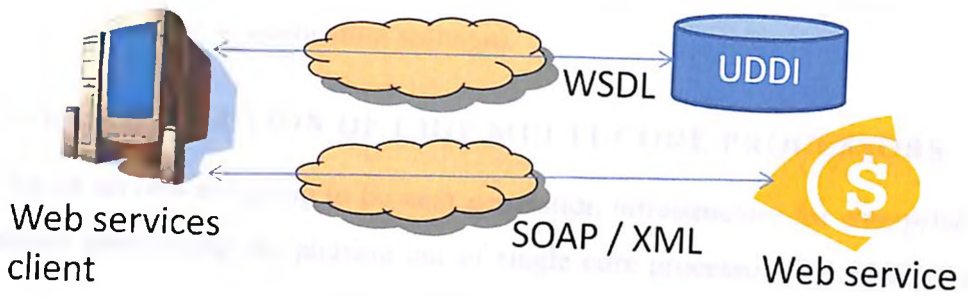


Figure 1-1 Simple view of Web services

Table 1-1 McKinsey view of evolving SaaS based business models

Services	SW at premises	SW as a service	Sources of savings with software as a service
Customization, integration	108	72	Reduced deployment time, limited customization, self-service through on-boarding scripts
Basic infrastructure testing, deployment	54	0	Does not require infrastructure and application testing
Application infrastructure testing and deployment	30	0	Same as above
Training	101	34	Lowers training requirements through simpler user interfaces, self-training service capabilities
Management, customization of business process change	94	0	Does not require ongoing business process change management, vendors monitors customer usage to enhance the offering
Data centre facilities rental, operations; security, compliance; monitoring of incident resolution;	750	0	Includes vendor's cost to serve in subscription price (ongoing operations, back-end hardware and software)
User licenses, subscription, maintenance	480	1500	Includes vendor's cost to serve in subscription price (ongoing operations, back-end hardware and software)
Unscheduled downtimes	308	0	Provides generally 99.99% availability
Unused licenses	92	0	Users are added as needed, hence there is a reduction in licensing cost.
Total costs	2298	1640	

Therefore, the web services based service oriented architecture is a growing trend for designing and building enterprise application software.

1.3 HARDWARE - EVOLUTION OF CHIP MULTI-CORE PROCESSORS

Multi-core based servers are going to be next generation infrastructure for enterprises [97], as hardware vendors announcing the phasing out of single core processor [93], [152] from the product lines.

Single core processor contains a single core. The performance of the single core processor is continuously improved through Instruction Level parallelism (ILP) [189] and Thread Level Parallelism (TLP) [81], [225].

- ILP is achieved by increasing the number of instructions that can be issued on every cycle from the processor's instruction queue, and by increasing the clock frequency of the processor.
- TLP is achieved by executing multiple threads in parallel.

Further improvements in hardware design, such as Reduced Instruction Set Processing, pipelining and superscalar multi-instruction issue techniques provided the base parallelism.

Efficient compilation techniques provided further parallelization by scheduling instructions and relevant data that enhanced the further speed of execution. Continuous pursuit for ILP and increase in number of transistors inside the processor required higher power and powerful cooling technologies to dissipate the heat generated during the processing. This increase in power requirements and limitations of cooling technologies are not desirable for the enterprise infrastructure. This led to the revolutionary chip multi processor design (CMP) [169], and also called as Multi-core processor design. CMP uses multi threading technology, by exploiting the thread level parallelism (TLP) that exists in the application workloads. CMP [107] as shown in Figure 1-2 contains two or more cores in a single chip or central processing Unit (CPU). CMP provides four major areas of enhancements over single chip processors, which include:

1. Memory fetch latency - Speed of CPU is much higher than main memory. CPU is stalled when data or instructions are being fetched from memory and this wait time is called "memory fetch latency". On single chip processors larger on-chip caches and ILP are used to reduce this latency to some extent. CMP uses TLP to address this

fetch latency [225]. When a thread is waiting for memory fetch, the core switches to execution of another thread, thereby reducing the memory fetch latency.

2. Execution flow: CMPs optimize the execution flow to get more work per clock cycle. This is achieved by using various techniques such as pipe lining, branch prediction, even reordering, out-of-order, and multiple threads are used.
3. Power consumption and die size: As more and more transistors are put into single chip processor, power consumption and hence heat dissipation increases exponentially. Power puts a practical limit on how reliable and fast CPUs can get, as well as the number of CPUs that can be packed in a unit space. CMP is power efficient as more processor work share the same power for delivering more processing performance; CMP is space efficient as there are more processors in a single die with a very marginal increase in the die size.
4. Processor complexity: More transistors in a single chip processor means, the design, debugging and verification of chip become very complex. In comparison, CMP almost provides the linear integration of relatively smaller cores, which are easy to design, debug and verify.

Various configurations of Multi-core design as shown in Figure 1-3 and Figure 1-4 are available in commercial implementations [29]. Some of these are Intel Pentium Hyper-threading, dual core, Intel quad core, AMD, IBM Power4, Sun Niagara, Tilera, nVidia. Advanced designs of Multi-core processors as shown Figure 1-5 (n) are in prototype stages in research labs such as Intel Research Labs [121], Tilera product development lab [221], [102], AMD Research Labs [8], and nVidia Research [168]. These configurations [153] vary in the how the core is structured internally to improve the instruction processing speed and how the memory is shared to reduce the latency of access from L1 cache, L2 cache and the memory buses. The concept introducing multi threading logic is included within each of the cores along with multiple registers, one per thread, which provides one level of optimization. This is seen in Hyper-threading [45] introduced by Intel which allowed two threads to share the same core. This concept enabled the core to process the instructions, while the other threads are assembling data from L1 cache. Currently there are multiple models of Multi-core chip designs from various commercial and research organizations.

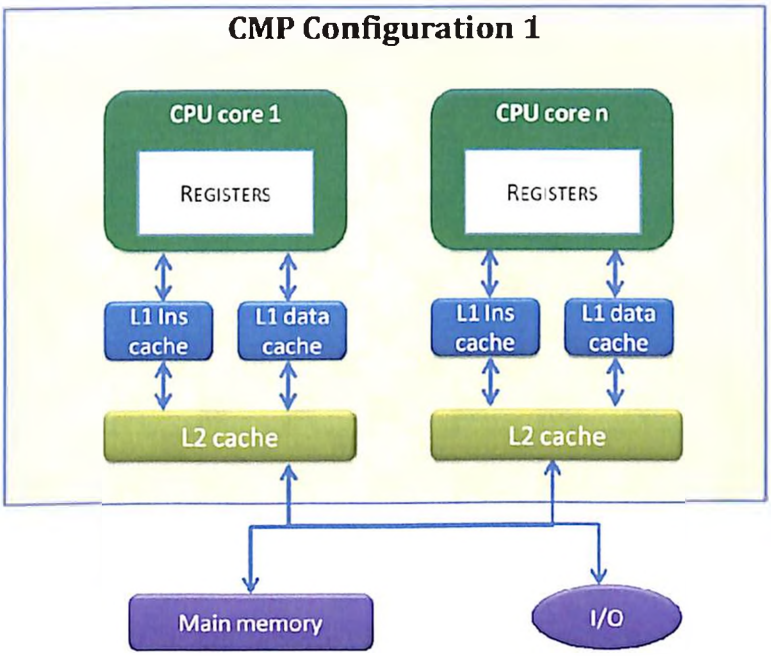


Figure 1-2 Configuration of Multi-Core Processor - Dedicated L1 & L2 caches for each core

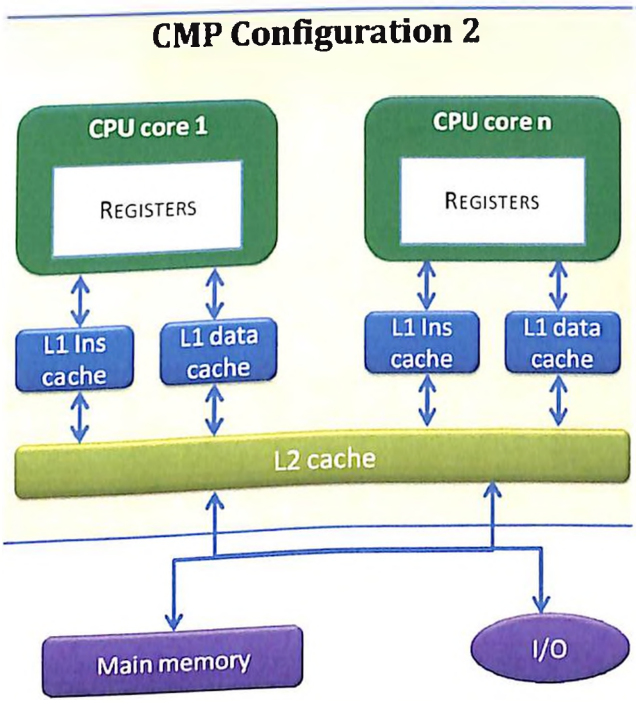


Figure 1-3 Multi-core processor configuration - dedicated L1 cache & shared L2 cache

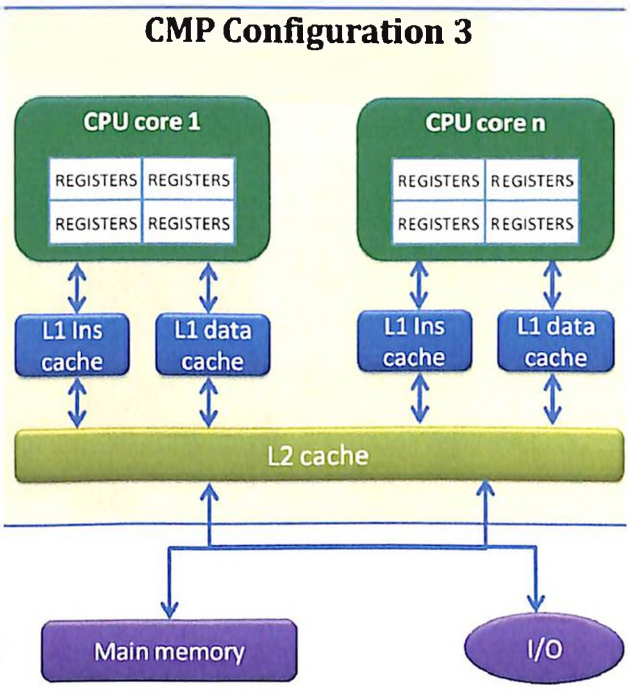


Figure 1-4 Multi-core configuration - Multiple registers per core, dedicated L1 & shared L2 cache.

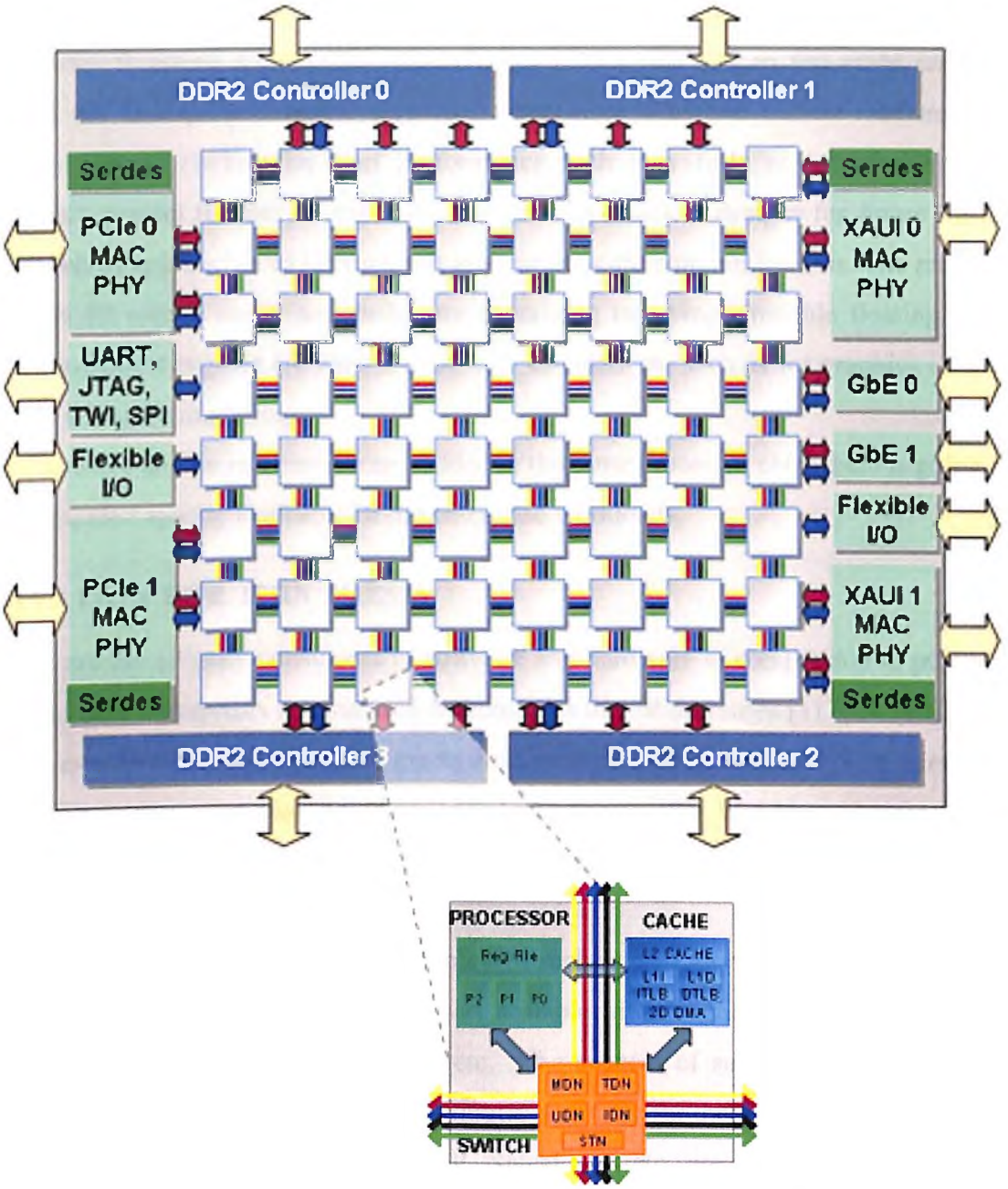


Figure 1-5 Intel Teraflop Chip – 80 cores on a single chip

One such model developed as a research prototype for commercial enterprise needs at Intel is called Teraflops Research Chip [122]. This chip is Intel's first silicon tera-scale research prototype. It is the first programmable chip to deliver more than one trillion mathematical calculations per second (1 Teraflops) of performance with very little power consumption. Teraflops research project focuses on exploring new, energy-efficient designs for future Multi-core chips, as well as approaches to interconnect and core-to-core communications. The research chip implements 80 simple cores, and each core containing two programmable floating point engines. Floating point engines are used for accurate calculations, such as for graphics as well as financial and scientific modelling.

In future, as more and more cores are embedded in the same processor chip, there is going to be abundant compute capacity available for the enterprise workloads.

1.4 MOTIVATION FOR THIS THESIS

Enterprises are the primary consumers of software and hardware in comparison to personal use. Well established enterprises use matured technologies like Mainframes [117] to fulfil their business needs; examples of such enterprise giants are Credit card companies like Visa, FirstData etc., flight booking companies like Amadeus, retail companies like Wal-Mart, Target in the USA, stock exchanges like National Stock Exchange in India, Mobile system providers like Airtel, Reliance, etc.

Recent generation enterprises use new technologies such as web services, Multi-core servers etc. to build the cost-effective, but state-of-the-art scalable infrastructure. Examples of such enterprises are Google, Amazon, EBay, Yahoo etc. The success of such enterprises prompts established enterprises to evaluate the new technologies seriously for their internal consumption. Platform vendors such as IBM, Microsoft, SAP, SalesForce etc. are motivated to push the advancements in technology to these enterprises. Therefore, it is evident that both the web services and the Multi-core platforms will be reaching out to all the enterprises rapidly in near future. This is first motivation to look into adoption of web services and Multi-core platforms in building enterprise class applications.

Web services architectures are inherently designed for short-lived atomic operations [4]. Web services based architectures are perfectly suited for all recent generation applications like Yahoo directory; Google Search; Facebook, a social collaboration platform; these companies

built their system architecture from ground-up, and those architectures are predominantly stateless. Furthermore large number of threads at the Multi-core reasonably suit short-lived operations that are independent and atomic. However, enterprise system architectures are designed for long-lived operations, [58], and for transactional [47] operations. Enterprise application architectures may not exhibit inherent parallelism to leverage the power of large number of threads. This is second motivation to investigate the possibilities of enabling parallelism by grouping the operations with common context and data. This grouping in turn could influence coherency among the operations within the application, so that these applications are able to use of threads effectively while executing in Multi-core systems.

Web services architecture support continuous evolution of functionality by separating the interface from implementation [90], so changes to web-service architectures at the implementation is relatively an easy job. Enterprise systems have been maturing over business models that exist for multiple decades, and those business models are not rapidly changing. For example, application use cases like opening an account in the bank, paying the insurance etc. have not changed for decades. Enterprises wait for the technologies to mature before they adopt those technologies [210], [92] preferably without disturbing the existing applications. This is third motivation to investigate the architectural enhancements that are needed to leverage the power of Multi-core processors without rewriting the existing business applications.

Matured enterprise software applications are designed for single chip processor architecture. In single chip processor architecture, the processor is the main constraint and it is time-shared across enterprise workloads. Multi-core processors remove this constraint by providing abundant number of compute threads. This makes the other sub-systems of the server such as memory, I/O, network speed etc. as the new constraints [167] of the system design. The effect of this shift of constraints in the server due to arrival of Multi-core and the associated impact on enterprise application design and deployment is the fourth motivation.

Finally, current operating system scheduling techniques [209] assume the stateful behaviour of the application, namely temporal and spatial locality characteristics [206] of the application. Web services based applications are inherently stateless and hence, do not guarantee temporal and spatial locality characteristics [254] of the application. Additionally, there is a limited provision available in the current architectures to transfer the application context to operating system kernel so that operating system can intelligently use that information for scheduling [32].

Our interest was to find out ways in which we can pass the application context thereby enabling the operating system to do intelligent scheduling.

This dissertation is motivated by the need for an architectural framework that can leverage the hardware computing power inherently available in Multi-core processor to serve the high performance web services based enterprise applications.

1.5 SCOPE OF THE THESIS:

Enterprises are motivated to adopt Multi-core servers for improved computing performance in their infrastructure, especially when one looks at the benchmarks [89], [166] demonstrated by Multi-core servers for few specific applications. This interest to bring Multi-core to enterprise opens up number of interesting research questions [186]. In this section we summarize the current challenges that we culled out as most important for our research. In the remainder of this section, we present our approach that was designed to address these challenges.

Deployment configuration could be another effective mechanism for Multi-core adoption other than rewriting the existing code for parallelism: Writing a parallel code is the way ahead for writing a new green field application targeted for Multi-core server. Significant amount of work is available in the literature [20], [28], [56], [84], [123], [124], [133], [155], [222] to guide on how to write a new application specifically to leverage Multi-core. However, in the enterprises significant amounts of capital investments and labour have already gone into developing the applications and these applications are in production. Enterprises are averse to infrastructural risk. In retrospect the massive software change management that enterprises needed, for making a date change to address the Y2K problem a decade ago validates the rationale behind this resistance to bring Multi-core to their infrastructure. Therefore, rewriting the entire application just for deploying Multi-core is a challenge in the enterprises. If we understand and address these challenges of the enterprise such as not requiring rewriting the application, then the adoption of Multi-core would become more prevalent in the enterprise. We identified special requirements that need to be fulfilled for adopting Multi-core in the enterprise, and these requirements are addressed in section 2.1.

Three non-invasive mechanisms are developed with an idea – "observe at the operating system level and accordingly change application behaviour without modifying the application", so that the application can utilize threads supplied by Multi-core. They are

- Control the number of requests that are dispatched to right server based on threshold configuration set for that particular setup. The admission control is done at the higher layers of application level with the knowledge of current workload.
- Modify the deployment parameters of the run time so that operating system is processing desired application process in context. This is done at the middleware platform level.
- Configure the cores efficiently with right size. This is done at the operating system / hardware level.

We introduce the concept of Core Scheduler which does the admission control and use the concept of Connection-Orientation to do contextual dispatch, and the concept of Infrastructure Awareness to identify appropriate core configuration and deployment configuration parameters relevant to the operating environment.

Stateful web services need differentiated processing design when compared to Stateless web services

Enterprise applications are wrapped into stateful and stateless web services. If the web service is stateless, then a thread is spun off and these web services can be grouped to run in parallel. There are architecture models such as SEDA (Staged Event Driven Architecture) address the ability of leveraging Multi-core for stateless web services. In the case of stateful web service, the context of the user needs to be preserved between requests. In this scenario, spinning a new thread to process an incoming client request ends up in Cache misses at the operating system level, due to temporal nature of the data. Instrumentation mechanisms are used to monitor the "Cache misses". These "Cache misses" are correlated with system deployment parameters. It is possible to use this correlation to modify the deployment configurations to reduce the Cache misses. Using the number of "Cache misses" as the base parameter, two additional parameters are derived namely Core Configuration Effectiveness and Intra Process Efficiency. These two parameters are together used to understand the behaviour of the "application in context" for specific deployment configuration setting. With iterations and sampling at both ends of the baseline, the optimum settings are established. These settings are fed back to Core Scheduler through ServiceMap, so that the client request is contextually connected to the web service application, which is tuned for optimum Multi-core utilization.

Structured architectural approach required for Multi-core adoption

Enterprise applications are typically designed using well matured architecture paradigm. Noteworthy examples include Microsoft Doc-View design paradigm followed by Model-View-Controller paradigm in separating user experience from persistent data store. Similarly, Client-Server paradigm led to three-tier and n-tier architecture in separating how to partition the application processing.

Software platforms and applications are designed for the time-sharing constraints of single-core. Multi-core, while addressing the time-sharing constraints through abundant threads brings in new space-sharing constraints. The adoption of new design constructs in the application specific to Multi-core is going to be evolving. Therefore, we need a bridge strategy that can embrace the adoption of Multi-core, while supporting the existing applications that are already developed for single core.

We have developed a new architecture framework called COAF (Connection-Oriented Architecture Framework) with six modules that can support the evolution of Multi-core for enterprise applications. The need for observing the application behaviour and setting up thresholds at the time of deployment as an architectural need would provide the mid-way path for Multi-core adoption. Analogically, this is similar to how every web application is designed in three tier architecture style as presentation, application and database tiers. While designing the web service application, the following questions need to be considered:

- What are the parameters to be observed?
- How to correlate the kernel behaviour to the application deployment?
- How to contextualize the processing workload with current deployment?
- How to schedule the workload at the application level?
- What is the right deployment configuration for optimum utilization?
- How to define optimum utilization in the context of Multi-core?

Well defined methodology is required for correlating the application behaviour to the deployment configuration.

Enterprises use correlation techniques in different ways; from correlating the product campaign to sales increase; the enhanced warehousing process to reduction in inventory cost; similarly, enterprises employ full time administrators to monitor and manage the technology

infrastructure. These system administrators are specialists on the platforms they monitor; examples of such administrators are Linux administrator, Database administrator, SAP administrator etc. These administrators could be equipped with tools with methodologies for correlating the system behaviour against deployment configurations with ease with minimal training. We have demonstrated an approach containing following steps to correlate the kernel behaviour with application deployment configuration.

- Identify the parameter to observe and insert the appropriate probes to monitor and collect the system logs for that parameter. "Cache miss" is used as the parameter to observe at the operating system level using DTrace probes. This is part of the COAF architecture - Behaviour Information module.
- Separate the outliers from the log and reconstruct the events that happened at the operating system.
- Identify the deployment parameter that can affect the observed parameter. We used the "number of MySQL threads" as the deployment setting.
- Establish the relationship between the "deployment configuration setting" versus the observed parameter. In this step we separated out "relevant application runs" from "overall runs", and arrived at the "cluster of relevant application runs". Using Pearson's correlation method, we established the relationship between the "number of clusters" against "core configurations" for a specific application configuration values (e.g., number of client threads).
- Arrive at the threshold values
- Store the "deployment threshold values" against the "particular core configuration" for a specific application use case or groups of use case. We have the configuration server as part of the architecture that can house all these lessons for current and future use.

Architectural abstraction needs to embrace new innovations in memory to accelerate the Multi-core adoption.

During our experiments one of the observations that we made is the effect of main memory to the compute infrastructure. Interesting innovations in provisioning the large size main memory such as FusionIO, Virident etc. could facilitate temporal locality characteristics of data, which is very important in the enterprise context, where most of the master data is read-only. To

embrace similar such new innovations, we have modelled both hardware and software as WS-Resource using Web Services Resource Framework, so that both hardware and software can be inspected through a web-service call. This would allow us to change the implementation of software and hardware and inspect their configurations without modifying the enterprise application.

1.5.1 Hypothesis

The following hypothesis is evaluated in this thesis:

i. It is possible to modify the performance of web service based enterprise applications from the application layer.

ii. It is possible to leverage Multi-core threading capabilities by changing deployment configurations and core configurations.

iii. It is possible to alter the performance of an application, by observing the kernel and hardware parameters, without any modification to the operating system.

iv. It is possible to affect the performance of a generic application in Multi-core system without changing the operating system configurations.

v. It is possible to effect changes in the performance of enterprise applications without changing either the enterprise application binaries or enterprise application source.

vi. It is possible to effectively contain and distribute enterprise apps amongst core to achieve performance through core symmetricity and core affinity.

In evaluating this hypothesis, we make the following specific contributions.

i. Introduce a set of design considerations based on the combination of enterprise experience and hardware/software vendor recommendations, and a resultant framework that could demonstrate the concepts through a prototype.

ii. Combine the ideas of performance improvement at hardware and software layer as highlighted in the literature; so as to secure and control the performance of the application through the input deployment parameters. The effect of key input parameters is studied and extended to individually observe the effect of temporal and spatial locality for any type of core configuration in the prototype.

iii. Implement a reference model of COAF, based on identified architectural considerations and configure the implementation using standard tool sets provided by the respective vendors. A typical test bed based on COAF framework is presented.

iv. Show that there are foundational primitives viz. the kernel level probe input and the application deployment parameters are used to address the performance along the lines prescribed by both the Hardware and the Software vendors, and derive a metric that can determine an overall performance of the application.

v. Evaluate the scalability and flexibility of the test bed by varying the configuration parameters for different hardware setups that are typical of a modern enterprise infrastructure; and to demonstrate that the architecture not only scales, but also manages performance across cores, significantly.

1.5.2 Outside the scope of research

Our interest in this research is to identify an architecture framework that can influence and facilitate the adoption of Multi-core in the enterprise. In this section we want to highlight the areas of work, where our focus is limited to using them rather than enhancing them.

Our experimental setup is based on Niagara server and Niagara provides mechanisms to partition and configure the number of cores to create various core configurations using "Solaris Zones and Containers". This is detailed in Appendix III. We assume that in future, every operating system manufacturer will provide mechanisms similar to "Solaris Zones and Containers" to configure the number of cores into a server. While we saw the impact of core configuration in various combinations such as symmetric, asymmetric combinations etc., the discovery of correct core configuration itself is a separate research area for administrators and configuration managers.

Similarly, the scope of our thesis is limited to demonstrating the impact of deployment configuration parameters on Multi-core utilization, but, finding the actual threshold parameters and the associated values is the subject matter for new research.

We have demonstrated the power of using an in-production instrumentation framework [231] of DTrace. The actual evaluation of all the instrumentation mechanisms and the best practice suggestions on how to use those tool sets are outside the scope of this work. While we

enhanced some of the DTrace probes to suit our experimental setup, we suggest the tools user to consult extensive literature available specific to their own setup.

Through "Connection-Orientation" and "Infrastructure Awareness" concepts, we highlighted the effect of passing application context to the operating system, through deployment parameters. We believe this approach will motivate further research in enhancing the operating system to receive the application context in more formal ways. Designing a new operating system or modifying an existing operating system scheduler etc. is outside the scope of this thesis and it is a subject matter for new research for operating system research community.

We tried to capture the consequences of other subsystems of a Multi-core server, such as main memory size etc., but identifying or designing the actual memory setup is not the scope of this thesis.

For conducting experiments, various tool sets and frameworks are used to demonstrate the architectural concepts of COAF. Improvements to those tool sets and frameworks are outside the scope of this thesis.

1.6 CONNECTION-ORIENTED ARCHITECTURE FRAMEWORK

We propose a Connection-Oriented architecture framework (COAF) for enterprise class web services on Multi-core (COAF). Two architectural concepts namely "Connection-Orientation" and "Infrastructure Awareness" and six modules are identified as the fundamental building blocks for designing the next generation enterprise applications based on web services that are hosted on Multi-core architecture.

COAF aims to fit the stateless web services that are served in request-response cycle into stateful needs of enterprise applications. The web service requests are grouped in such a way they can be scheduled to the same core so that temporal and spatial characteristics of the application can be leveraged. Additionally, the total requests that are dispatched at any point in time to the processing server are controlled by the Core Scheduler at the admission time to pre-set threshold values. Thus, the "Connection-Orientation" is established between the requester of the service and the appropriate processing core that is best suitable for processing the service request.

COAF leverages the feedback from the "Infrastructure Awareness" gained from using in-production instrumentation mechanism. This information is used for setting thresholds values for various combinations of infrastructures for the optimal performance of Multi-core servers.

The six modules are Core Scheduler module, Threshold Configuration module, Resource provisioning module, Behaviour Information module, System Management module, and Notification Framework module.

Core Scheduler module maintains two objects called SchedulerState and ServiceMap for each kind of web service. ServiceMap maintains "where to dispatch information", SchedulerState maintains "current state of the server" and the scheduling policies for the Scheduler. Threshold Configuration module maintains the threshold values for configurations for various components in the system such as processing cores, memory caches and I/O ports etc. These threshold values are the combination of run time information obtained empirically from instrumentation of the system and design time information obtained from the application developer before deploying the application. To arrive at the threshold values the system parameters are analysed and correlated for various kinds of deployment configurations of the system. "Cache miss" is used as the observed parameter at the operating system level. We designed two derived parameters namely "Core Configuration Efficiency" and "Intra Process Efficiency" that are based on Cache miss. Using these two derived parameters, we study the results of our experiment to get the threshold values. These threshold values are then stored in Configuration server, so that the application platform is deployed with these new threshold parameters. Once the application is bootstrapped with these threshold values, the system can be monitored for the performance of the core. These monitored values are compared against the preset threshold values and the process is repeated until optimal performance is obtained at the level of processing core for the current enterprise application deployment. Thus the knowledge of operating system scheduler behaviour and the contextual knowledge of the application are combined to achieve the optimum utilization of Multi-core compute power.

We have demonstrated the above approach through five phase experiments as outlined in section 5.1. The experiments are designed in such a way, that lessons from previous phase are incorporated into the next phase so that experiment focus is more targeted on Multi-core thread utilization. We have observed interesting results during our experiments, such as

- In one such configuration setting, few cores are not used at all, even though operating system had the opportunity to use them. Even though there were 32 cores available, operating system scheduled only 6 cores for scheduling workloads.
- Similarly, 17 cores configuration performed well compared to 16 cores configuration, even though 17 cores configuration has overheads associated with "interconnect fetch latency". This is because operating systems scheduled its own management activities in 17^{th} core, so that the application job is scheduled in the remaining 16 cores.

1.6.1 Organization of the Thesis

This thesis is organized into six chapters. In this Chapter 1, we have already discussed two evolving trends related to web services and Multi-core platform and discussed the motivations behind this dissertation and finally our proposal for COAF as the new architecture framework for Multi-core platforms. In Chapter 2, we explore the related research work and the underlying technologies that are relevant to our thesis. We begin with discussing how to achieve parallelism through Multi-core at the hardware level, and then go on to discuss how recent generation architectures exploit the parallelism in the system. In Chapter 3, we discuss the Connection-Oriented Architecture Framework on Multi-core (COAF) and its design features such as modularity, and maintainability. The six modules of COAF are explored in the remainder of the chapter. In Chapter 4, we have detailed the implementation of each of the six modules along with their internal structure and components used to implement them. In Chapter 5, we discussed the five phase approach and conducted experiments on COAF based setup for testing the spatio-temporal characteristics, across four deployment models, namely (1) traditional Client-Server model (2) Client-Server model, both client and server running at same process space, (3) Client-Server model, where the client and server are located in different process space, and (4) Client-Server model for differentiated loads. Summary of the results for each of the phases are discussed. In Chapter 6, we conclude our thesis with its contributions and benefits. Finally in Chapter 7 we highlight the opportunities and scope available for future work on this research.

2 LITERATURE SURVEY

In this chapter, we develop the case for COAF by studying the previous research and approaches published towards utilizing the processing power provided by Multi-core for processing enterprise application workloads. First we discuss the properties of the enterprise class applications and the properties of web services thereby able to identify the requirements for COAF design. Next, we look into programming models and architectural approaches that are available in the literature related to one or more aspects of the Multi-core servers and web service based applications. We analyse two fundamental programming approaches adopted for exploiting the parallelism available in the Multi-core systems using both thread based concurrency and event based concurrency. Then, we start to look in to various architectural frameworks that address large volume processing, either at the hardware or at the software level. From the hardware perspective, it is both general purpose and specialized architectures (both at chip level and at the overall system level innovations). These hardware architectures mostly focused on adding more cores to the processor and addressed necessary changes required at the level of cache. From the software perspective, it is both web service architectures and specialized architectures that exploit inherent parallelism that exists in data and instructions. Next, we discuss various deployment frameworks and their impact on application design. In the summary section, we identify the gaps that need to be addressed by the COAF framework in order to marry the power of web services and Multi-core threads.

2.1 INTRODUCTION TO ENTERPRISE APPLICATIONS

We studied and analysed the external and internal properties of various enterprise applications broadly under three categories, namely (a) Client-Server model deployed on the internet but within the enterprise (examples include Microsoft Exchange and Outlook [162], time sheeting system, operations management system, invoice and billing system, customer relationship management system, project management systems and payroll processing system); (b) Client-Server model deployed over the web (examples include trading applications like ETRADE [72], search applications like ASK.com [21], large ecommerce applications like ebay.com [67], ERP systems like SAP Netweaver [197], Intuit TurboTax system [126], Quicken and personal finance systems); (c) Client-Server model deployed as Software as Service model

over cloud. (Examples include Customer relationship management systems like Salesforce.com [195], video management systems [232] like Blockbuster online [30], application life cycle management systems using Serena Mashups [200], security management systems like Symantec Safeweb [215], ecommerce systems like Amazon Web services[6], benefit compensation systems like EquityEdgeOnline [73]). These studies revealed a broad set of requirements for the web services based enterprise application. The following three categories of properties are considered as key requirements for enterprise application architecture on Multi-core. They are,

1. General properties of the enterprise application – these properties look at various general aspects of the enterprise application such as application composition, management aspects, etc.
2. Spatio-temporal properties of the enterprise application – these properties look at internal behaviour of the application.
3. Stateful properties of the enterprise application – these properties look at the deployment nature of the enterprise application.

2.1.1 General properties of the enterprise application

General properties section looks into functional and non-functional aspects of the application and thus helps to arrive at the requirements designing a new architecture framework.

2.1.1.1 Property 1 – Well defined business processes

Enterprise applications are the software systems that are used to run the day to day operations of the business efficiently. Typically these applications are general purpose or custom written for a specific business problem. These applications could be internal to the enterprise such as employee facing application or external to the enterprise such as customer facing or supplier facing applications. These systems are either executable software that can be loaded on desktop or a large system that is loaded on a server on the network that can be physically accessible by a desktop. Examples of such enterprise applications include email management systems such as Outlook, office administration systems such as Word, Excel, or large enterprise human resource system such as SAP, Oracle applications, Peoplesoft, Customer support systems like Siebel, Business Intelligence and reporting systems like Business Objects etc.

Enterprise systems operate in Client-Server model [176]. The client portion of the software called front-end software typically operates at the desktop. This could be a rich application like windows based application/applet etc., or could be a lightweight browser based application. The server portion of the software called back-end typically runs on a single server or on a cluster of multiple servers. These servers themselves could be a single processor machine or a multiple processor machine such as mainframes.

In an enterprise, there are a fixed set of well-defined users with a well-defined access and authentication policies, typically segmented across different functions of the enterprise. For example, human resource personnel will have the access to business functions of the human resources department, whereas the procurement department will have the access to business functions related to purchase and order management. Figure 2-1 shows a typical enterprise application footprint, along with the key business processes of the application represented as web service components.

Thus in summary, these enterprise applications are typically exposed as well defined sets of business processes (as web services), supporting fixed set of users; typically employees and suppliers.

2.1.1.2 Property 2 – Sequential process dependencies

A typical business process is a sequential process [63], requiring the logic to be performed in a particular order so as to preserve the integrity of the operation as a whole. For example, an invoice cannot be processed, till the inventory is received at the warehouse. Hence, processing needs to happen in a particular order like in the above example, lookup the purchase order master table for validating the purchase order number that is present in the invoice, validating the receipt of the item mentioned in the invoice at the warehouse, receipt of the quality assurance certificate for that item, and then the invoice can be processed.

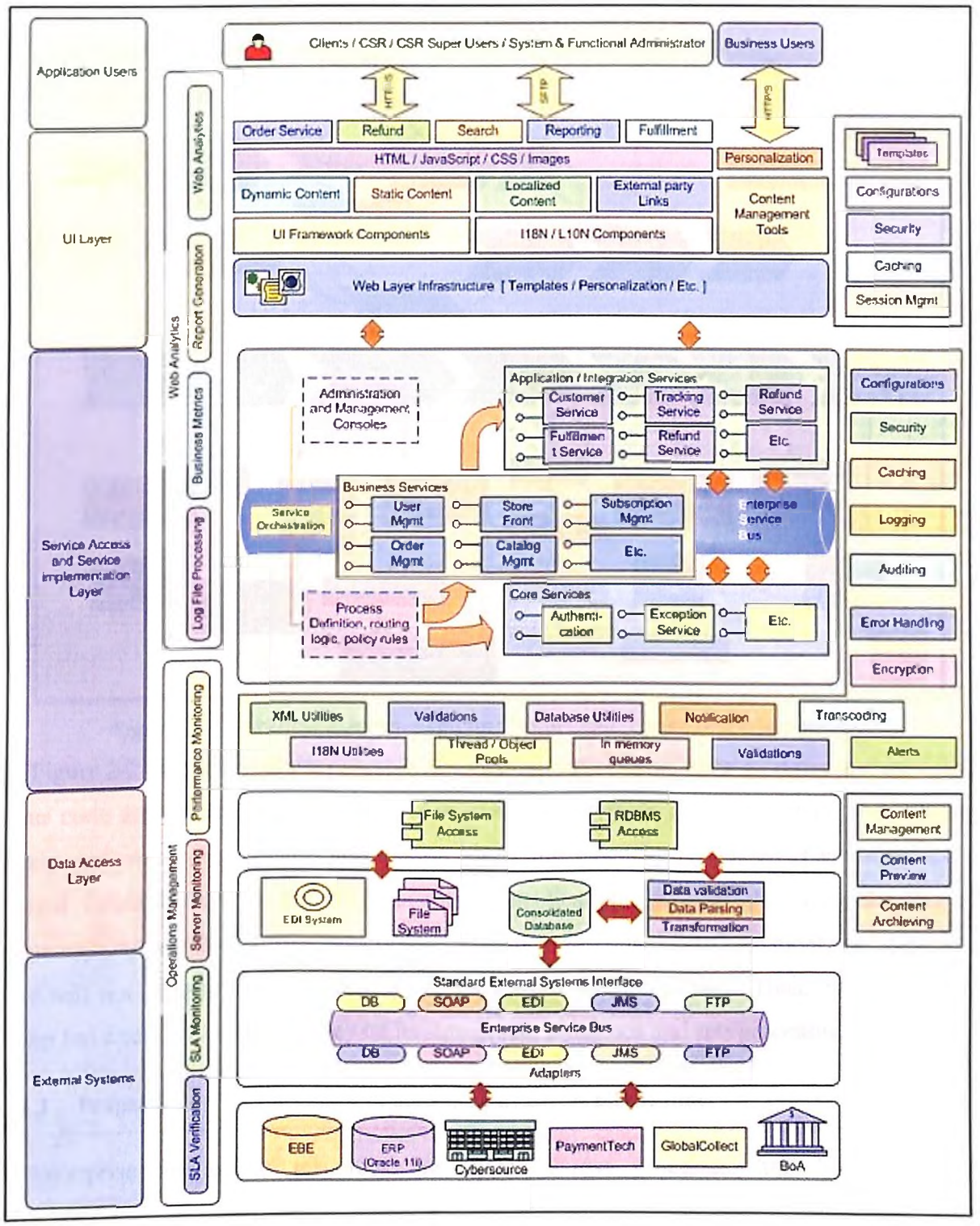


Figure 2-1 Architecture diagram of a typical enterprise application

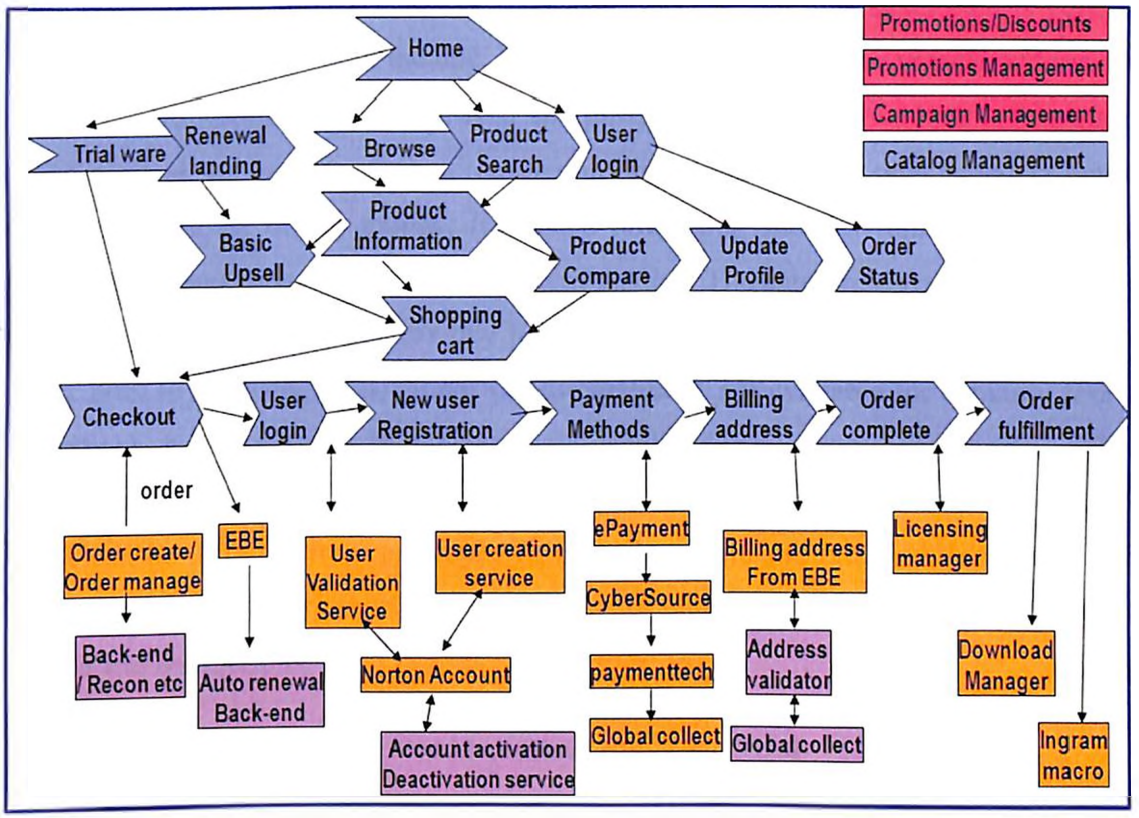


Figure 2-2 Workflow diagram depicting "long lived" dependent components

Figure 2-2 shows a real life example of shopping cart system. Here modules that are orange colour code are external dependencies. For example, payment is dependent on four systems, namely epayment - where the information about the payment is entered, Cybersource - an external validation system for credit history authentication, paymentech - for collecting the payments, Globalcollect - the reconciliation system for a consolidated payment etc. Similarly, order will not happen until payment and billing steps are completed. Thus, every enterprise system has a sequential dependency on its subsequent processes and sub-processes.

2.1.1.3 Property 3 – Process and Data need to be available all the time

Enterprise application resources are shared across processes, and hence need be continuously available during work hours with two assumptions. They are: (1) Mostly same user works on the system – this means, user logs into the application from the start of the day and stays connected till he logs out during the end of the day, and generally one log-in happens throughout the day. Most of the time same user logs in from the same desktop (or socket or

network access point) and this desktop typically resides within the enterprise network. (2) Most of the data is created and used within the enterprise network, and this same data is generally available and consumed throughout the day. For example, purchase order is created and the same purchase order is viewed, reviewed, approved and pushed to suppliers. Two kinds of operations [26] happen most of the time; (a) Online transaction processing (OLTP) such as ledger entry at the bank for withdrawal of the money; (b) Batch processing such as running reconciliation reports for invoices and collection during the end of the day. The enterprise systems are expected to be available throughout the business day for doing both kinds of operations. For example, when a user goes to the bank, bank cannot say the system is not available for banking. For the above operations, resources such as databases are usually shared across components. But the data size between two similar transactions can vary (for example, number of medical bills for an employee A can be different from employee B, even the results are received through the same logic.) Therefore, the enterprise architecture needs to support both spatially and temporally coherent data and instructions.

2.1.1.4 Property 4 – Enterprise workload behaviour can be modelled

In general for every business function there are two kinds of users; standard user and admin user; Standard enterprise user logs on to system to do day to day business functions during the standard business working hours say 9.00 am to 6.00 pm. Similarly batch processes and report generation processes run on a specific duration of the day – typically in the evening. Admin user logs on to the system to do administration functions of the system, which is typically non-business working hours such as night time from 11.00 pm to 4.00 am. Business functions in well-defined cycle, and hence the supporting processes need to function in that cycle. Examples include, Payroll happens in a particular frequency (one in a month, or once in fortnight or once in a week), time sheet submissions happen once in fortnight mostly on the 15th day, trade reconciliation and settlement happens at the end of stock market closure and gets completed before the start of the market next day, the finance audit report gets prepared at the end of quarter, tax filing happens once in quarter and for common public it happens around April in the USA, and in June / July in India. Hence the frequency of the workload arrival can be predictable and modelled [32].

2.1.1.5 Property 5 – Enterprise Users stay connected always – User sessions never expire

The business critical and monotonous nature of these enterprise applications required very high productive user experience for these systems; hence these systems are designed as connection-oriented systems. The enterprise application architecture assumes that there is a private connection needed between the user desktop client system and the server system; the server is expected to maintain a separate session and the usage context about the user who logged in till the user voluntarily logged out of the system. This means, once the user is logged in, there is a dedicated space is provided in the memory and this space is available in time for that user throughout the session till the user logs out of the system. This memory space is used to store all the interactions between the user and the system. This memory space provides session context for the application about the user.

2.1.1.6 Property 6 – Enterprise Infrastructure contains matured and heterogeneous technologies

Enterprise class applications are business critical systems and hence lot of care is taken in selecting the infrastructure of the enterprise to reduce the risk of loss of business continuity. These systems are extremely hardened against all known exceptions and hence would require huge investments. The change management processes are extremely complex and cumbersome and hence every new infrastructure or design needs to be compliant with standard architectural blue prints, so that in the event of failure, alternate equipment or software can be readily pluggable to continue running the infrastructure. Over time these enterprise systems mature and stay with the enterprise, which results in heterogeneous infrastructure. A typical infrastructure for a large enterprise would contain multiple hardware configurations, operating systems, application servers and data bases. Instead of making the infrastructure homogenous, typically an integration effort is undertaken to manage the heterogeneous infrastructure. This is a unique behaviour of the enterprise infrastructure design compared to custom designed application for a specific business need.

In summary, the properties of enterprise application are:

- Well defined sets of business processes and hence web services
- Workload behaviour can be modelled
- Structured and Sequential process dependencies

- Process and data need to be available throughout
- User stays connected and hence the session never expires.
- Infrastructure contains matured and heterogeneous technologies

Having identified the functional requirements, the next step is to understand the internal implementation of the workload processing and there are two important properties namely spatial locality and temporal locality [206] that derives the workload processing on enterprise servers. The next section addresses the literature around these properties.

2.1.2 Spatio-temporal locality properties of enterprise application

The operations between the client and server are generally synchronous, which means that when the client requests the server for data, the server sends the response immediately to the client with minimal or no verification about the client application. Server assumes that it is connected with the client all the time, since both client and server are deployed on the secured closed network. This behavioural assumption on enterprise applications led to the design and development of processing systems. Thus the temporal and spatial locality becomes the key assumption for the architecture design and system design as seen in Figure 2-3. If some data is referenced, then there is a high probability that it could be referenced again in the near future. This is called temporal locality [206]. In the Figure 2-3, the data accessed by CPU in cycle 1 is stored for future use, and this data is fed in cycle 3 from cache thus saving the fetch time from memory. If some data is referenced, then there is a high probability that data next to this data could be referenced in the near future. This is called spatial locality [206]. In the Figure 2-3, when CPU is fetching the data for cycle 1, the neighbourhood data is also fetched and stored in cache for future use. When the neighbourhood data is needed in cycle 2, it is fed from cache thus saving the fetch time from memory. These two definitions are key requisites for an enterprise application.

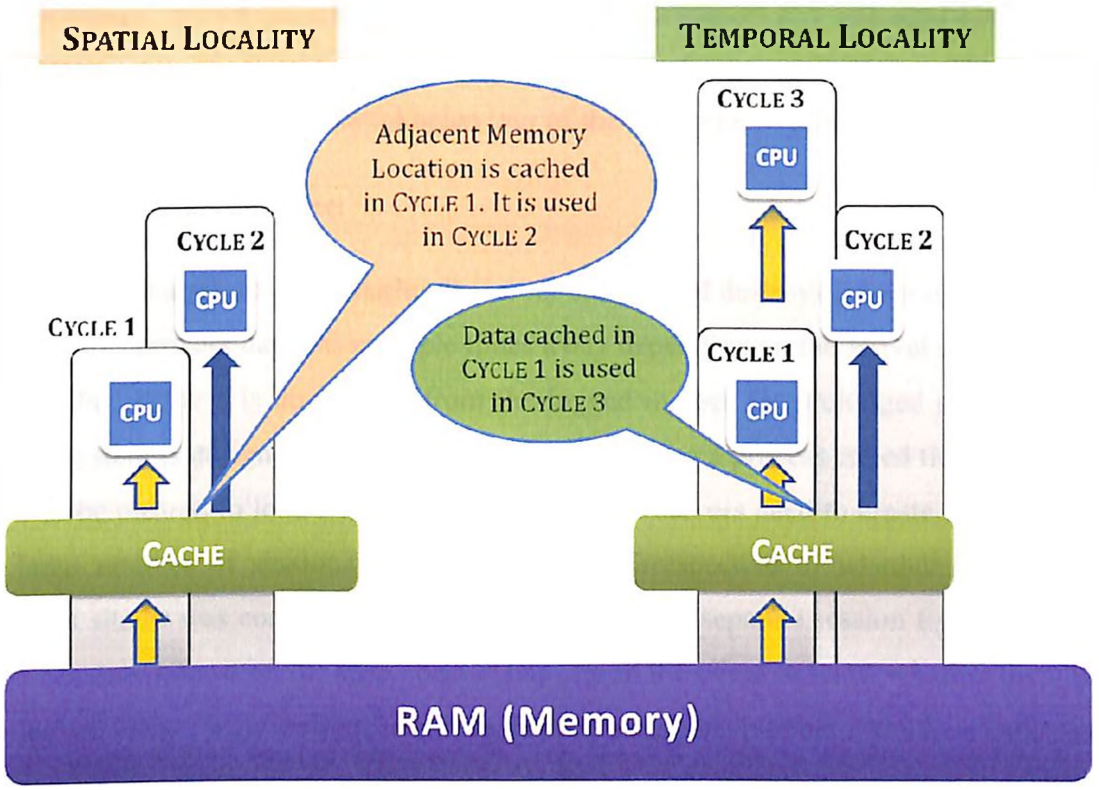


Figure 2-3 Spatial and Temporal Locality

2.1.2.1 Enterprise applications need "look up" data and hence caching:

Enterprise applications reference master data often. This need reflects the spatial and temporal locality characteristics of the data. Algorithms and techniques were developed to keep the copies of the data (that is used most often or used more recently) adjacent to the processing core. This concept of keeping the data nearer to the processing zone is called caching. At the hardware layer, multi-level memory cache is developed such as register, L1 cache, L2 cache, Main memory, etc. to keep the data and instructions closer to the processing core. At the application layer, similar concept of caching is developed, and the examples include Memcached [157], Jcache [128] and Terracotta [220], etc.

2.1.3 Stateful properties of the enterprise application on stateless web

Enterprise applications are stateful. They operate in a Client-Server model, hosted on a closed secured network. This closed secured network allows application to keep the context of the user between operations. The web provided cheap networking medium and easily adaptable protocol standards. Initial web applications were static information portals using stateless HTTP

GET/POST protocol. HTTP GET/POST protocol became permanent and web applications were designed stateless. Google Search is the example of such stateless application. This stateless nature of the web is anti-thesis to stateful behaviour of the enterprise application.

2.1.3.1 Standard Web architecture:

In the web based application scenario, the server creates and destroys the session objects for every user, and this process happens multiple times a day depending on the arrival characteristics of these requests. If there is no activity from the logged in user for prolonged periods of the time, then the system is designed to destroy the session by using a process called timeout and the caches have to be cleared to load a new data. This means servers need to create, maintain and destroy a large number of sessions for the same user, irrespective of whether the user did anything in that site or was completely inactive. Maintaining a separate session for a non-active user is an expensive process on the web. Server has to poll the client to know whether the client is still an active client. This polling needs to happen over the internet, which is both time consuming and expensive process. So the web based system encourages the stateless or connectionless design pattern in the application architecture. Connectionless design pattern also suits the web, as the web is fundamentally connectionless or otherwise stateless. The notion of Connection-Orientation in web is established through various workarounds such as HTTP persistent connections, cookies, URL rewrites, hidden fields in the form, TCP streaming etc. This stateless design of the web makes the caching assumptions difficult, as the arrival patterns of user requests are not predictable and the data becomes stale very quickly if the data is transactional in nature.

2.2 EXISTING FRAMEWORKS DESIGNED FOR PARALLELISM

In this section various approaches and frameworks that are built to extract the parallelism available in the processors at the hardware and software level are discussed. First, we look for inherent parallelism to exploit processing speed. Research and advancements in processors in general relied on two forms of parallelism: instruction-level parallelism (ILP) and thread-level parallelism (TLP). Multiple techniques, algorithms [25], [106] and specialized hardware architectures [78] are developed to identify and exploit the ILP and TLP in programs. Wide-issue superscalar processors [66] exploit ILP by executing multiple instructions in a single cycle.

Multiprocessors exploit TLP by executing different threads in parallel on different processors. Natively parallel applications share the cache and branch-prediction, and compete for identical functional units, thus exploit ILP. Non-parallel applications compete for cache and branch prediction hardware [213] and execute independently, thus exploit TLP. Tullsen et al proposed simultaneous multi threading processor [225] that can use thread-level parallelism and instruction-level parallelism interchangeably. However, in this design, multiple threads cause inter-thread interference in the caches and place greater demands on the memory system, thus increasing average memory latencies and thereby decreasing the overall system performance. There are two common multithreading programming models namely: Thread programming and event driven programming.

2.2.1.1 Thread programming model

In thread programming model [27], a separate thread of control is created for tasks, such as for each network connection, input device, user, or other appropriate entity, the flow of control can be explicitly given. In thread programming key issue is the contention between threads for the resources and hence the latencies created due to thread context switching. Synchronization operations, such as locks and semaphores, are used to protect shared resources and data structures. Blocking and Spinning are two mechanisms used in thread synchronization. Spinning is a waiting mechanism with which the waiting thread continuously checks for occurrence of a synchronization event. Blocking is an alternate waiting mechanism with which the OS suspends the waiting thread and schedules another thread to execute. Both these threading mechanisms have implications on the processing of enterprise applications, as the blocking delays the user experience, while spinning can result in sub-optimal performance.

There are two classes of algorithms developed to reduce the overheads due to thread memory models.

- Majority of the work is focused on lightweight threading for shared-memory multiprocessors such as Stackless Python [208] for creating micro-threads, lazy allocation techniques such as Lazy threads [95], Lazy Task Creation [163][44] , and Stack Threads [218].

- Others focus on developing hybrid model comprising of event driven and threading such as cooperative threading, state threads and thread pools such as IBM Websphere [192], Weblogic etc.

There are other techniques that are built to monitor and fine tune the Operating system for effective thread performance such as idle spinning thread detection Li et al. [144].

2.2.1.2 Event driven programming model

In an event-driven programming model [180], a central event loop watches all external sources of data (e.g., network connections, input devices, and timers) and invokes call back functions to process each piece of data as it arrives. This model is prominently used in GUI toolkits and network programs like BIND family of DNS servers. In the event-driven program, the event loop is in control. When an event-driven program wants to perform an I/O operation such as reading some data from a network connection, it can't simply stop and wait for the data. Instead, it needs to set up an I/O request and then return to the event loop. When the data is available, the event loop will invoke an I/O completion callback function to process it. There are enhancements proposed to event driven programming model [239], architectural approaches [55], and custom implementations specific to Multi-core [113].

2.2.2 Hardware level parallelism exploitation relevant to Multi-core Architectures: Piranha, Hydra, Niagara, and Blue Gene/L

At the hardware level, many research prototypes and commercial implementations are built around threads, events and the combination of both. Among these prototypes, Compaq research prototype Piranha [25], Stanford Hydra [82], Sun Microsystems Niagara and IBM Blue Gene/L [2][91] have considered Multi-core as the fundamental design construct and built the compute architecture around that design. These designs are primarily developed for enterprise class commercial loads. In this section, the design constructs in these four prototypes are analysed.

2.2.2.1 Piranha

Piranha [25] is a chip multiprocessor, focused on improving the performance on commercial applications such as online transaction processing, decision support systems, which have characteristics such as large volumes of data, memory stall blocks, little ILP, abundant thread-level parallelism, very limited or no use of complex data types like floating point etc. At the

architecture level, hierarchically partitioned and replicated design, 8 single-issue in-order Alpha core processor, 1MB of private instruction and data cache, shared non-inclusive L2 cache with 8 banks, and each bank is 8-way set associative, 8 memory controllers. Interesting concepts in Piranha are the home and remote memory access to off-chip memory access, and cache coherence protocols and engines. Figure 2-4 depicts the design of Piranha. Additional interesting design concepts present in Piranha are Directory information maintained at node-level using hot-potato routing model, and Invalidation-based directory protocol.

2.2.2.2 Hydra

Stanford Hydra [106] targeted towards commercial workloads, uses a concept called thread-level speculation (TLS). TLS is the process of speculatively executing interdependent threads out-of-order, while appearing to have executed them in-order. Hydra converts processor instructions to sequenced threads, for example, loop iterations as separate threads and procedure calls as separate threads, so that threads can run in parallel. It uses hardware and software mechanisms to track inter-thread dependences. Corrective measures are taken for any violations by re-executing instructions with correct data. Hydra is four core design based on MIPS architecture, with private 16 KB of Data and instruction L1 caches, and shared 2MB L2 cache, with inclusive cache hierarchy. Figure 2-5 depicts the Hydra architecture.

The four processors share an on-chip, unified L2 write-back cache, and each processor executes a single thread. Each processor's L1 data cache is write-through. Other processors snoop the bus connecting the processors and the L2 cache. This is to permit data dependence violation detection. Dependences are tracked on a per-word basis, to eliminate the violations due to false sharing. Speculative result buffering is achieved by buffering speculative writes to the L2 cache in a group of 32- cache-line buffers, one for each processor. These buffers also monitor read requests made to the L2 cache. This allows them to forward data created by writes from less speculative processors to satisfy the requests of more speculative processors.

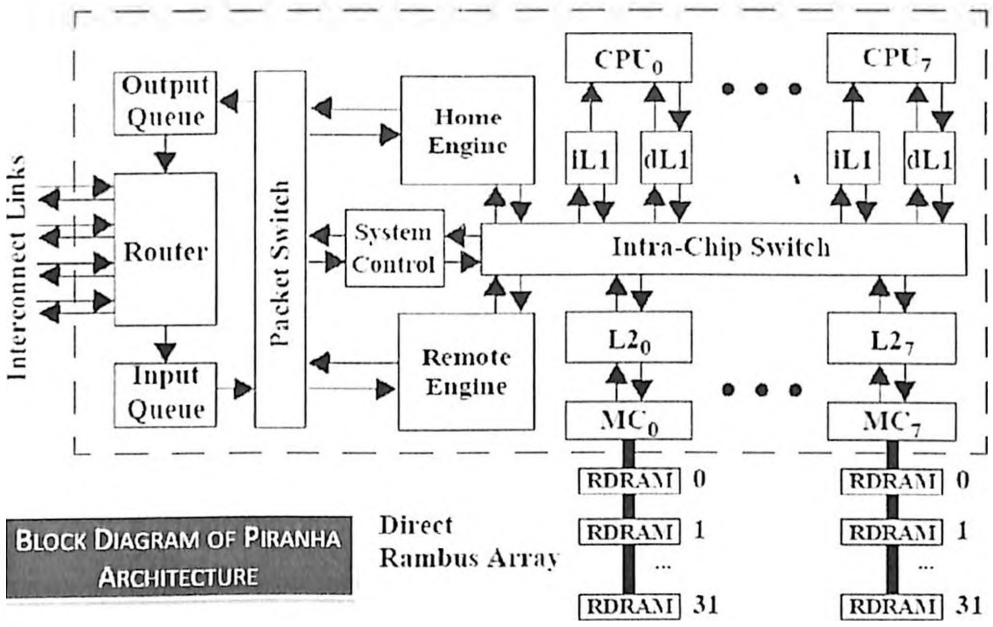


Figure 2-4 Piranha system level architecture (Piranha CMP [25])

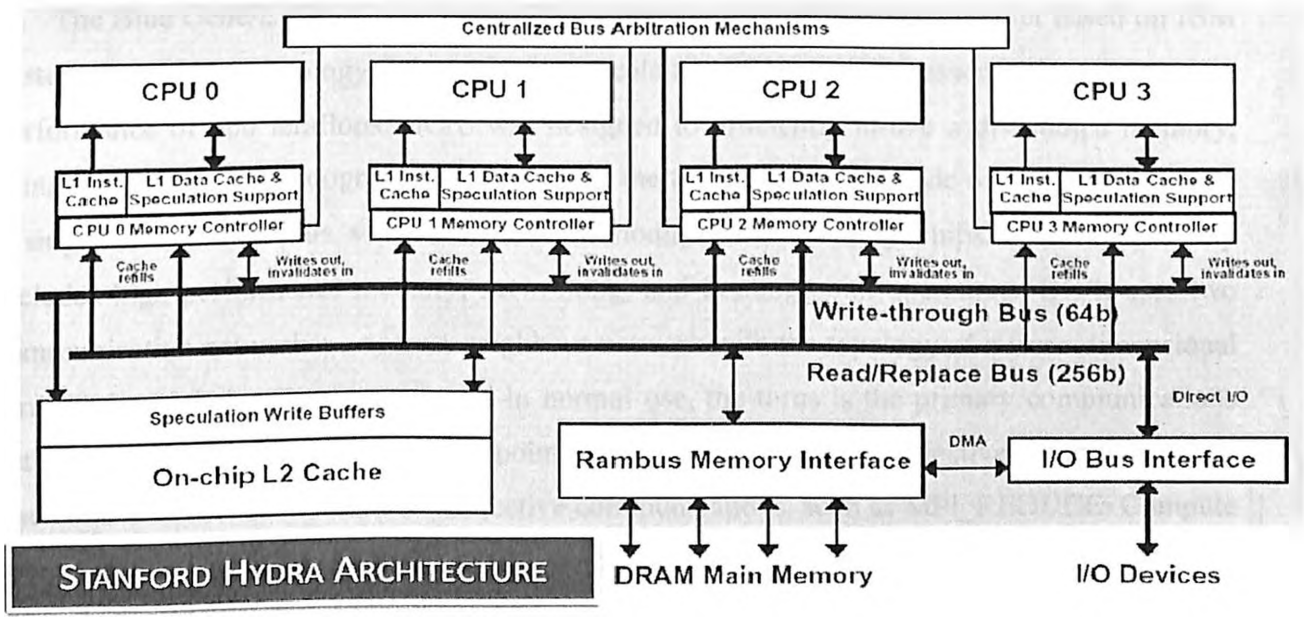


Figure 2-5 Hydra internal architecture (Stanford Hydra Project [106])

2.2.2.3 Niagara UltraSPARC T2 processor

Niagara [138], [96] is the code name for UltraSPARC T2 processor from Sun Research Labs. In T2, each core can run at speeds up to 1.4 GHz, and contains an 8 KB data cache and a 16 KB instruction cache. The block diagram and the internal architecture of Niagara are seen in Figure 2-6 and Figure 2-7 respectively. An eight bank, 4 MB unified L2 cache is shared by the eight cores. Each core contains dual pipelines, and a mechanism to switch between the four threads on each pipeline of a core such that a new thread is scheduled on the pipeline at each clock cycle in a round robin manner. Four dual channel Fully Buffered DIMM (FBDIMM) controllers provide a maximum memory configuration of 64 GB. The processor also includes eight floating point units (FPU), with a fully pipelined FPU per core. The eight cores, L2 cache, and memory controllers are connected via an on-chip crossbar interconnect. T2 processor is the second version of the T1 processor that is used in the experiments for this research.

2.2.2.4 IBM Blue Gene/L

The Blue Gene/L (BG/L) computer [2] is a massively parallel supercomputer based on IBM system-on-a-chip technology. It is designed to scale to 65,536 dual-processor nodes, with a peak performance of 360 teraflops. BG/L was designed to efficiently utilize a distributed memory, using message-passing programming model. All the functionality of a node was contained within a single ASIC chip plus some external commodity DDR memory chips. The functionality includes high performance memory, networking, and floating-point operations. BG/L uses two communication networks: a nearest-neighbour network with the topology of a three-dimensional torus and a global collective network. In normal use, the torus is the primary communications network and is used both for point-to-point and for many global or collective communications. The collective network is used for collective communications, such as MPI_REDUCE. Compute nodes on the BG/L are logically arranged into a 3D lattice, and the torus communications network provides physical links only between nearest neighbours in that lattice. Therefore, all communications between nodes must be routed in a manner that makes use of the available physical connections, and the cost of communications between nodes will vary depending on the distance between the nodes involved.

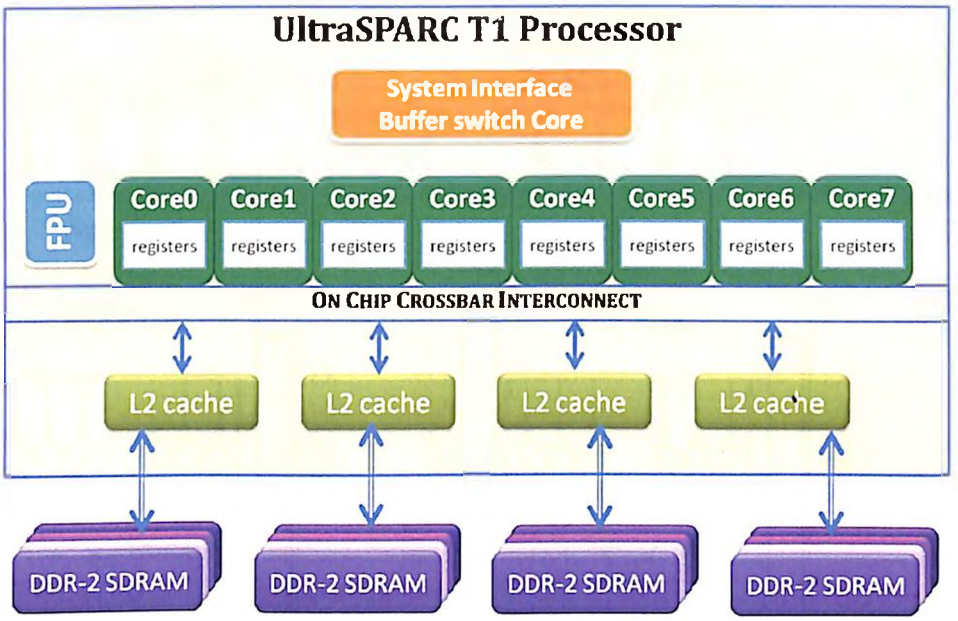


Figure 2-6 Block Level Diagram of 8 cores Niagara UltraSPARC T1

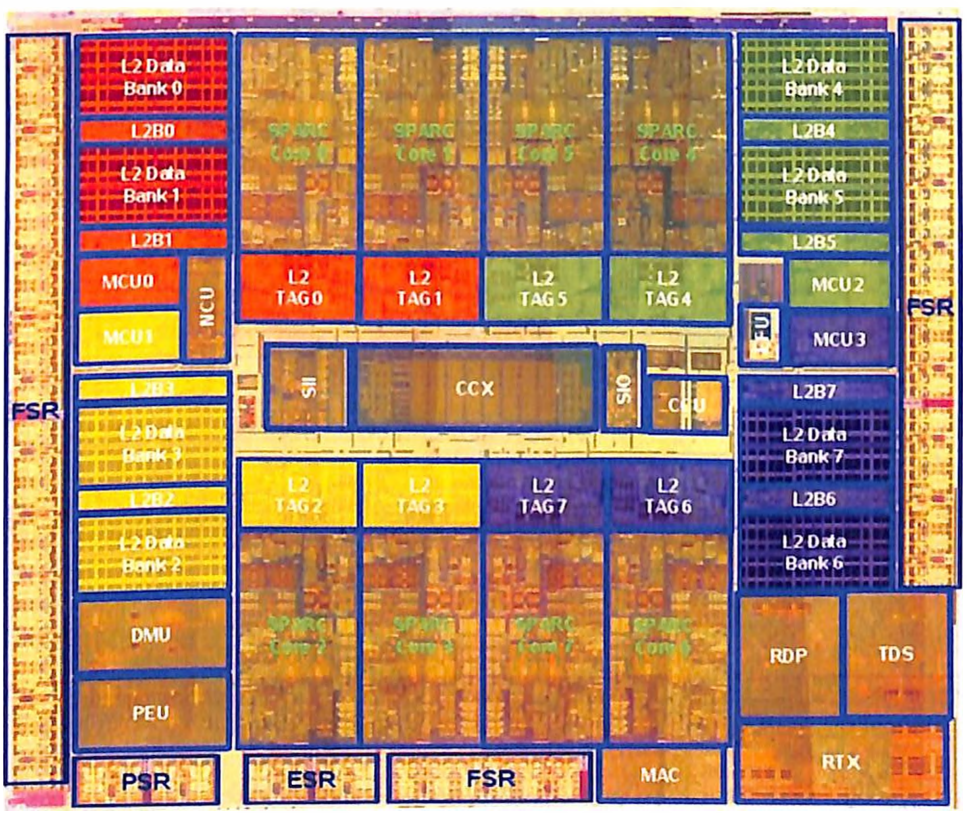


Figure 2-7 Niagara internal architecture

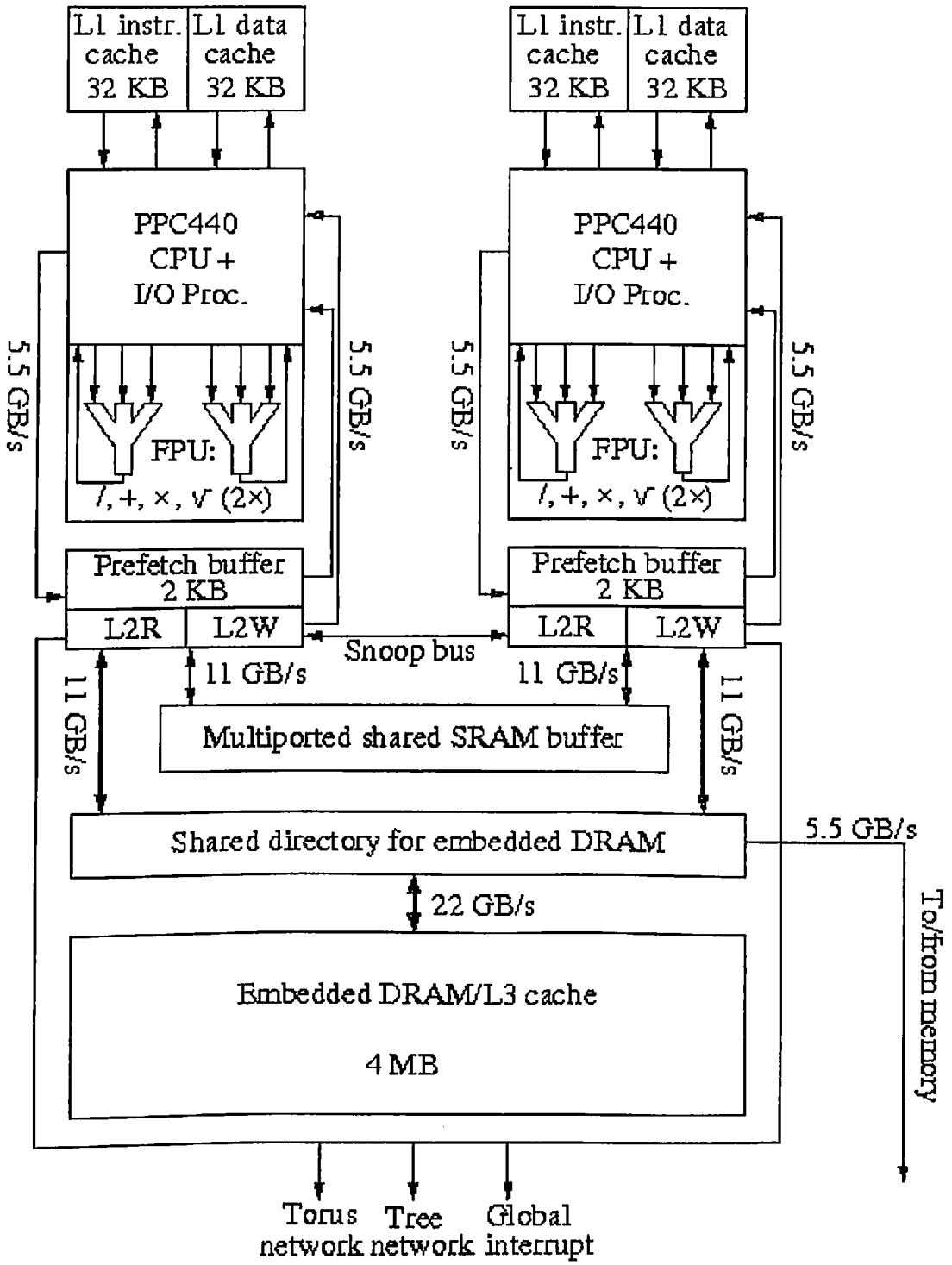


Figure 2-8 Information flow in Blue Gene/L architecture

BG/L used for niche, compute intensive applications such as molecular research, 3D Fast Fourier Transforms, and Interesting frameworks such as Blue Matter. This concept of using communications is leveraged in Multi-core architectures.

2.2.2.5 Summary of Hardware level Multi-core Architectures

Hardware architecture frameworks such as Piranha, and Hydra, expect explicit thread level parallelism in the application. Asynchronous stateless applications can directly leverage these prototypes to process the web service requests in parallel. Recent generation application servers and operating systems are multi-threaded, and they are designed to process any thread on "first-come-first-served" basis. "First-come-first-served" based design perfectly suits stateless asynchronous application requests such as web services. Similarly, compute intensive batch processes of certain kinds can leverage the power of these thread-intensive architectural styles.

2.2.3 Software level architectures: Click, StagedServer, SEDA, and REST

Software level architecture styles have been proposed to exploit parallelism that is inherently available in the stateless applications. In this section, the four prominent architecture styles - Click modular packet router [137], StagedServer [142], Staged Event Driven Architecture (SEDA) [239], and Representational State Transfer (REST) [70] [80] are studied.

2.2.3.1 Click

Click Packet Router [137] uses an architectural software construct called "elements". Router can have any number of input and output ports, and it performs simple routing computations. As shown in Figure 2-9, elements are linked with one another forming a connection mechanism, and the packet is routed through this connection on a single function call. Elements are implemented as separate component with its own state. Click uses a declarative language to model the router configuration. It uses a single thread for each processor and performs load balancing across threads. Click makes an assumption that modules have bounded processing times, which leads to a relatively static determination of resource-management policies. Click is single threaded and hence directly cannot take the advantage of Multi-core platform.

2.2.3.2 stagedServer

Larus et al. introduced a general programming paradigm called staged software servers [142][108] according to which the computation is divided into stages and there is a scheduler within each stage and implemented a prototype called StagedServer using Cohort Scheduling policy[142] as shown in Figure 2-10. This model aims to maximize processor cache locality, where service requests are grouped so that similar records are can run in same stage / batch. Using Cohort Scheduling, the processing is deferred until a similar service arrived for processing, thereby increasing code and data reuse across unrelated computations, that otherwise would have ended in cache conflicts or Cache misses. In the Staged computation model, threads are replaced by stages as the underlying software construct.

2.2.3.3 SEDA

SEDA [239] stands for Staged Event-Driven Architecture; bundles the concepts of events and threads using staged computation and thread pools as shown in Figure 2-11. Each stage is a self-contained application consisting of an event handler, an incoming event queue and a thread pool. Events are processed in batches to improve throughput. SEDA applies fine-grained admission control [39] at each stage to limit the rate at which events are accepted by stages. However, in SEDA there are no optimizations for memory hierarchy performance, which is the primary bottleneck for data-intensive applications (e.g., OLTP and DSS workloads).

2.2.3.4 REST

REST is an architectural style advocated mainly for the construction of web services. In REST, the software architecture is defined by a configuration of architectural elements-components, connectors, and data--constrained in their relationships in order to achieve a desired set of architectural properties. To create a REST service, the following questions need to be addressed:

1. What are the resources and in specific URIs?
2. What's the format or representation?
3. What methods are supported at each URI?
4. What status codes could be returned?

The focus of REST is to break the business logic into the types of resources and define it with its own URI. Representation of the resource could be HTML, XML, images or an audio. Resource may be accessed and/or its state can be modified using HTTP GET / POST methods. REST model is suitable for abstracting the implementation from its usage especially for CRUD (create, read, update and delete) over the web.

2.2.3.5 Software level architectural styles exploiting parallelism

Enterprise applications that exhibit the property "well-defined business processes" fits into domain of ClickRouter, StagedServer, SEDA, and REST architectural style constructs and principles. However, there are very few enterprise class application platform providers that inherently support these architecture styles in their platforms. Hence, if an external mechanism that can group and configure the web services in such a way that existing application server can leverage the multiple threads provided by Multi-core for processing the workload. The "Connection-Orientation" notion of COAF supports this ability for the enterprise application.

2.2.4 Data parallel frameworks: MapReduce, Hadoop, Dryad

Another approach proposed to exploit the parallelism is to design the architectural style that is based on the prior knowledge of the data processing behaviour of the application. Successful examples of this approach are Google MapReduce [59] [68] and [183], Yahoo Hadoop [11], and Microsoft Dryad [127]. These special purpose frameworks combine the data parallelism and multiple threads in the Multi-core servers [190]. The best practices of the MapReduce, Hadoop and Dryad architectures are discussed below.

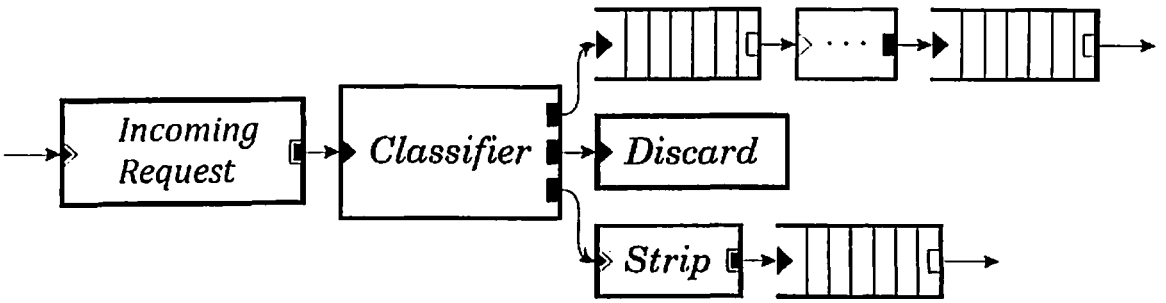


Figure 2-9 Click Modular Router

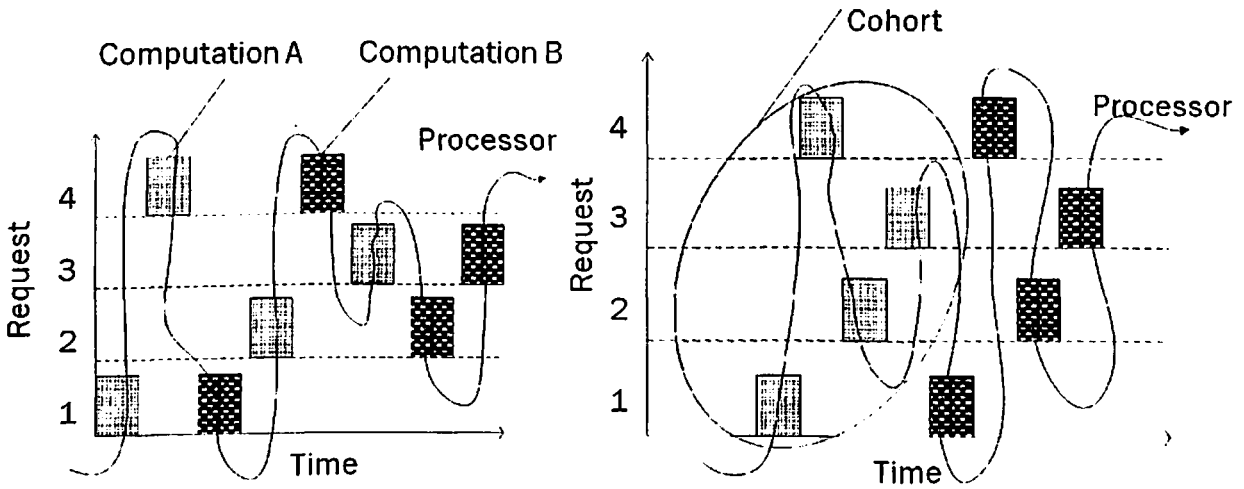


Figure 2-10 Cohort Scheduling

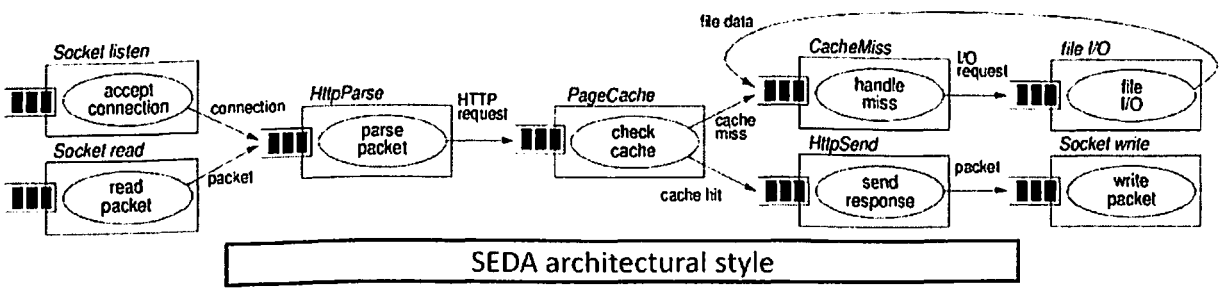


Figure 2-11 Stages in SEDA architecture

2.2.4.1 MapReduce

Google popularized the MapReduce paradigm, and uses it internally for processing terabytes of data across thousands of servers. MapReduce is both a programming model for generating/processing large data sets and an engine called "MapReduce system library" for taking care of parallelism, fault tolerance, data distribution and load balancing. MapReduce has two functional phases as shown in Figure 2-12; (1) Map() - process a key/value pair to generate an intermediate representation that are also key/value pairs; (2) Reduce() is a user-defined function. MapReduce library merge all intermediate values sharing the same key. Google MapReduce libraries along with Google File System and its storage mechanism called BigTable forms the complete implementation for leveraging the MapReduce paradigm.

2.2.4.2 Hadoop

Yahoo uses its own implementation of MapReduce paradigm called Hadoop, similar to Google MapReduce. Hadoop as shown in Figure 2-13, is a combination of distributed processing framework, distributed file system namely HDFS (Hadoop Distributed File System), and scheduler/Resource management libraries, distributed DB system called HBase. Once the servers are marked as Hadoop equipped servers and clustered as Hadoop clusters, invoking the MapReduce in one of the server will draft the other machines in the cluster to do MapReduce function.

2.2.4.3 Dryad

Microsoft research built the architecture and execution engine called Dryad (Distributed Data-Parallel Programs from Sequential Building Blocks) for processing coarse-grain data-parallel applications. This architectural style is depicted in Figure 2-14.

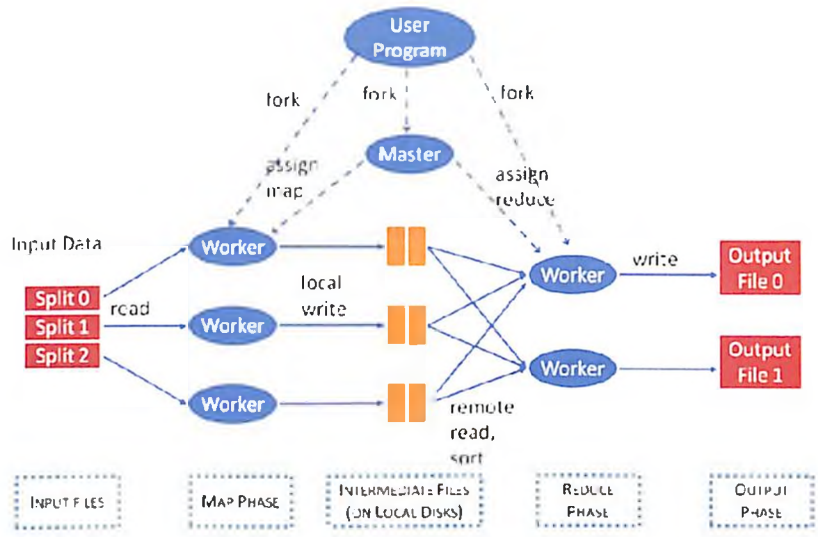


Figure 2-12 MapReduce Architecture (Google MapReduce)

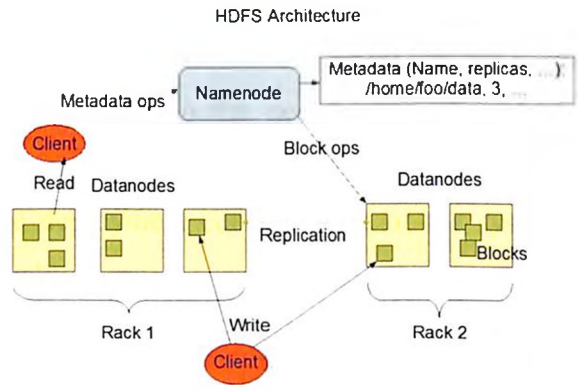


Figure 2-13 Hadoop Architecture Diagram (reference Yahoo Hadoop [11])

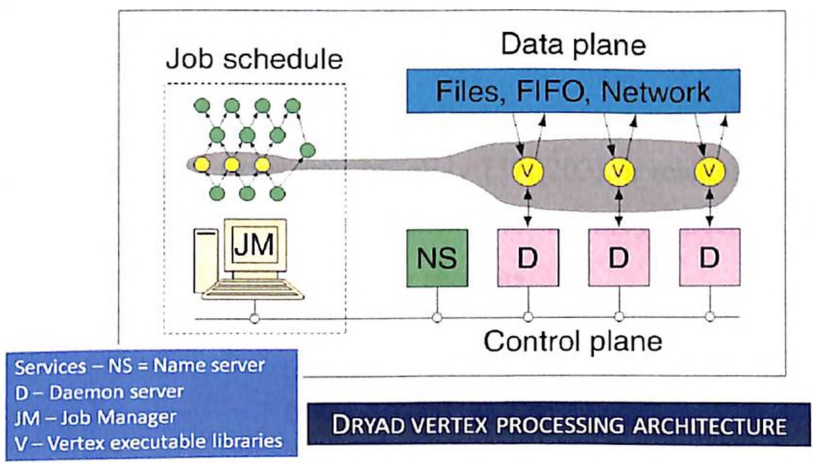


Figure 2-14 Microsoft Dryad architecture

Dryad is designed to scale across Multi-core computers, through small clusters of computers, through data centres with thousands of computers. Dryad introduces the concept of "dataflow graph", which is the combination of two foundation blocks called computational vertices and communication channels. Dryad runs the application by executing the vertices of this graph on a set of available computers.

2.2.4.4 Summary of Applications that are inherently data parallel

Similarly, frameworks such as MapReduce, Hadoop, and Dryad expect the applications to have data that can be processed in parallel. These architectural styles are suitable for certain classes of batch processing workloads where instruction and data parallelism exist inherently – e.g., account reconciliation, invoice creation etc., where we can analyse and capture the application behaviour at the design time and thereby setup the best configuration for processing. Understanding the application upfront in the deployment lifecycle is facilitated through the notion of "Infrastructure Awareness". This "Infrastructure Awareness" helps to identify the opportunities to enhance the deployment configuration of the system; thereby the underlying compute can be used effectively.

2.2.5 Application specific architecture frameworks

This section discusses four architectural implementations that are custom developed to solve a specific business or technology problem. These architectural constructs, concepts and best practices are extremely relevant for COAF. These four implementations aim to achieve instruction and data locality by segmenting the application into manageable parts. These are Layering and Tiering architecture developed by eBay [57][203], a relational query processing engine called QPipe, XML processing techniques and memory specific implementations specifically REDAC are discussed in the following sections.

2.2.5.1 Layering and Tiering architecture of eBay

Layering and Tiering is an architectural concept developed for the world's largest online marketplace, eBay. EBay has more than 250 million unique users. In this architecture, the layers provide software factoring and abstraction; tiers provide broad application partitioning and distributed processing. This aspect of separation of concerns in "layering and tiering" is significant to capture both design time knowledge and deployment time knowledge to optimally leverage the compute capability. The general architecture is based on low-level subsystems being used by high level application components. Specific components (or sets of components) are then exported as services (inbound, outbound and internal). Figure 2-15 illustrates the three architecture levels and their interdependence: The basic dependencies between the layers are that upper layers depend on lower layers. This is a strict enforcement. Each layer is partitioned across the tiers. An application can be thought of as being decomposed into the various layers and tiers and the application of the various systemic qualities.

The Figure 2-16 shows how the eBay application is partitioned into tiers and layers. Each layer has its own common code that is shared across the tiers. Layering provides the instruction locality and tiering provides the data locality, thus eBay architecture minimizes the performance overheads arise due to heavy network payload.

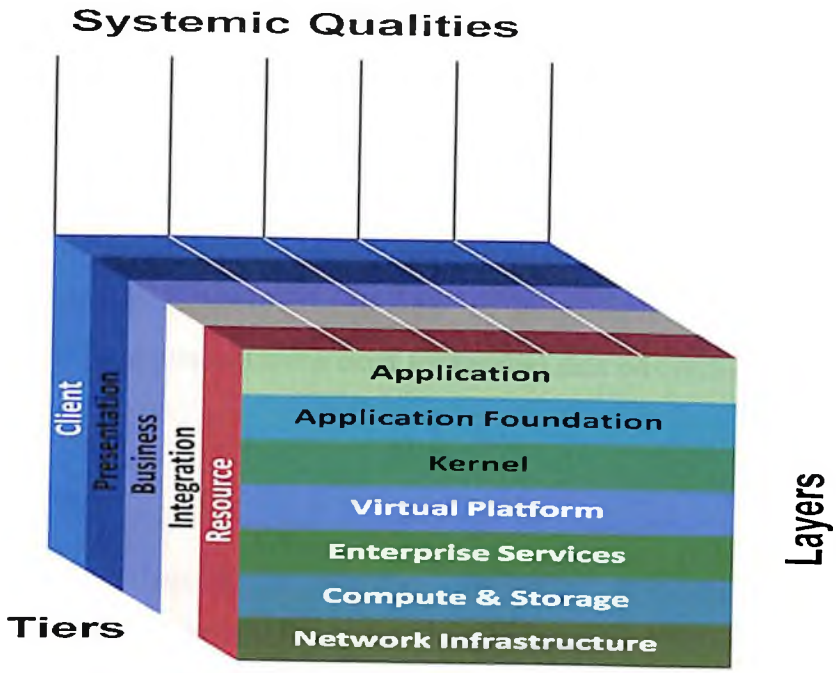


Figure 2-15 EBay layering and tiering architecture

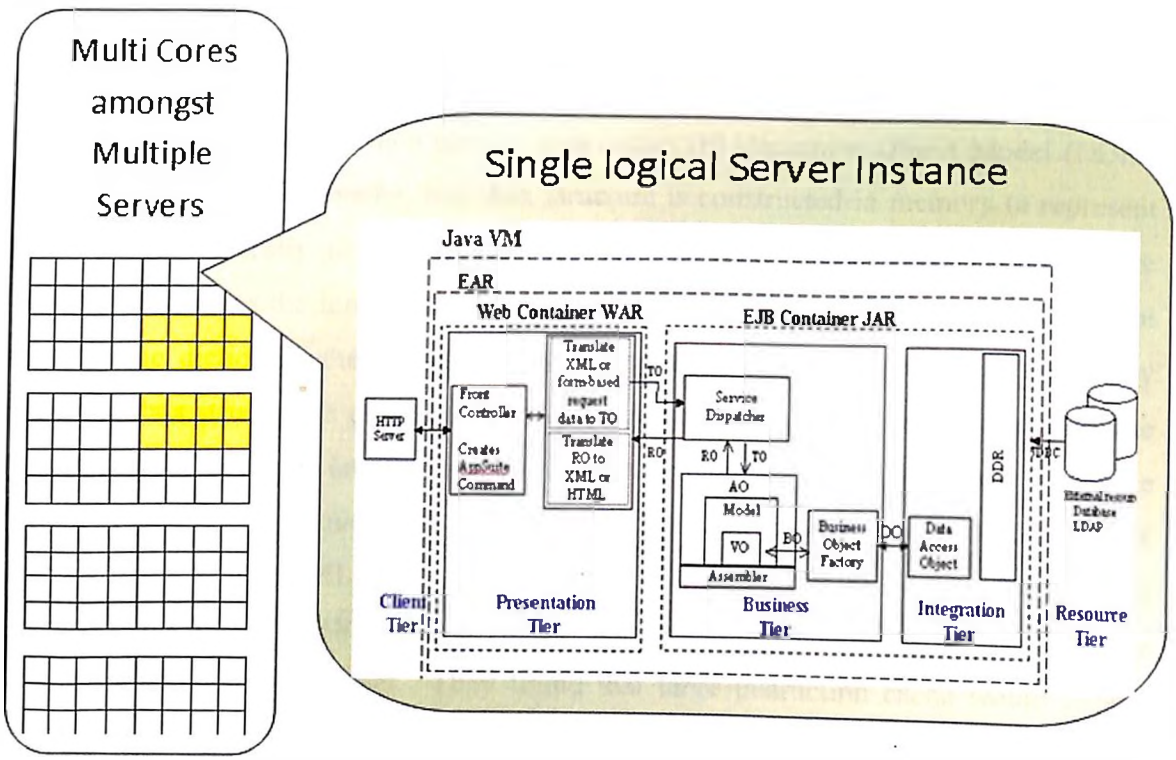


Figure 2-16 EBay application partitioning into tiers and layers

2.2.5.2 QPipe:

QPipe [109] is a query centric relational database system that follows the execution model of "one-operator, many queries" in contrary to traditional relational query engine designs that follows the "one-query, many-operators" model. QPipe leverages better instruction locality and data placement mechanisms, using query plans generated by its optimizer module. The execution engine evaluates queries independently of each other, by assigning one or more threads to each query. Similar commercial implementations exist for parallel data processing from vendors such as Teradata [219].

2.2.5.3 XML Processing:

XML [244] [19] has become an important declarative language for web services. Significant amount of research happened in enhancing the speed of XML processing. XML is processed in four fundamental styles [18], namely SAX, DOM, StAX, and VTD [141]. Out of these four styles, SAX and API are predominant in the web service implementations. (a) Simple API for XML (SAX) parsing: SAX-style parsing is event driven, where events are fired when each of these XML features is encountered during the parsing. The user defines a number of call back methods that will be called when these events occur; (b) Document Object Model (DOM) construction. - In DOM style parsing, tree data structure is constructed in memory to represent the XML document. Typically the memory used for DOM goes into three buckets namely (1) the memory buffer that stores the input data, (2) the tree structure that consists of various element nodes, and (3) the dictionary that stores special strings for the XML document. When fully constructed, the data structure is passed to the application, which can then traverse or store the tree. DOM-style parsing can be intuitive and convenient to integrate into applications. There are several XML processing techniques [62] evolved in the last decade software level and hardware level, ranging from binary XML [49], [249] to schema-specific parsing [48], [146], [69] to hardware acceleration [149], [182], [119], [256]. Li Zhao et al addressed the performance improvements at the Hardware level. They found that large instruction cache would greatly impact the performance of computing when compared to the impact of data cache. To increase the speed of the XML data parsing, they incorporated new instructions with special hardware support for frequently used operations.

2.2.5.4 Memory specific implementations - REDAC

There are several research and development efforts that address the advancements in large scale memories. These innovations in memory could influence the performance of Multi-core infrastructures. The interesting memory specific implementations using non-volatile flash at the hardware level are Solid State Disks (SSD) [116], FusionIO [88], Virident [231], and at the software level are transactional memory implementations [101][112][61][110]. One such implementation is REDAC memory server [94]. It is a set of lightweight mechanisms for distributed, asynchronous redundancy within a shared memory multiprocessor. REDAC provides scalable buffering for unchecked state updates, permitting the distribution of redundant execution across multiple nodes of a scalable shared-memory server. The REDAC mechanisms achieve high performance by enabling speculation across common serializing instructions and mitigating the effects of input incoherence.

2.2.5.5 Custom existing enterprise applications - Summary

The custom implementations such as eBay layering and tiering, relational query engine QPipe, XML processing engines, and memory specific REDAC emphasizes the need to have an inclusive and pluggable architecture that can borrow the best-of-the-breed implementations. WS-Resource standard provides the ability to abstract the resources thereby facilitates the pluggability of software and hardware implementation. Architectures mentioned in the literature addressed some of the properties of the enterprise application; however there are other properties that are not directly addressed by these architecture frameworks. These are stateful nature of the enterprise application, highly available system, matured and well-defined infrastructure governance processes that limit the ability to change the architecture and implementation at will.

2.3 DEPLOYMENT ARCHITECTURAL MODELS

The arrival of web services based middleware platforms promoted a "two life cycles" paradigm for the development and deployment of an enterprise application respectively. First life cycle is the development life cycle, is a set of engineering activities that are involved in converting the functional and non-functional requirements of the application into an executable code. These activities are requirement analysis, design, development, unit testing and system testing.

The second life cycle is the deployment life cycle, is a set of engineering activities, namely – *Packaging, Installing, Configuring, Activating, Deactivating, Maintaining* and *Uninstalling*, that are involved in making the executable code (a.k.a. software binary image) run on the hardware as described in Figure 2-17. The deployment life cycle could be simple or complex depending on number of components that constitute an application. The advent of component based software construction methods [176] provided the ability to separate the application functionality into multiple components. These components are then assembled to provide the functionality of the overall application. Then this application is deployed on top of the middleware platform such as J2EE application server, along with dependencies such as Apache Web server [12] and MySQL database [130] server.

Figure 2-18 shows the deployment view for a web services based application. As seen here, the deployment complexity gets enhanced, when the application needs to run on different middleware platforms and the middleware platform needs to run on different operating systems and the operating system needs to be configured for different server layouts. The configuration is defined as a set of parameters with associated values. Configuration values can be set either at server startup time or during the run time. At the application and middleware layer level, advanced component based architectures used deployment frameworks [60] that are developed using the declarative approach to define deployment information. The prominent examples of deployment frameworks include J2EE [1] [74], .Net assemblies [159], Fractal mode2l [245] [99], web services [46], and virtual appliances [214] on the cloud. Deployment Manager for Services Applications (DMSA) [229] presents an implementation and the meta-model based approach for the context-aware deployment of web service in constrained services execution environments. Grid5000 test bed uses Master-worker [35] paradigm for deployment model definitions. Another framework called DeployWare [82] comprising a meta model about deployment information, a virtual machine for deployment and graphical interface for managing the deployment.

Virtualization is used for deployment and configuration at the operating system level. There are three methods of virtualization at the operating system level [185] namely, hardware virtualization, para-virtualization and OS virtualization that are used for deploying the hardware resources for processing.

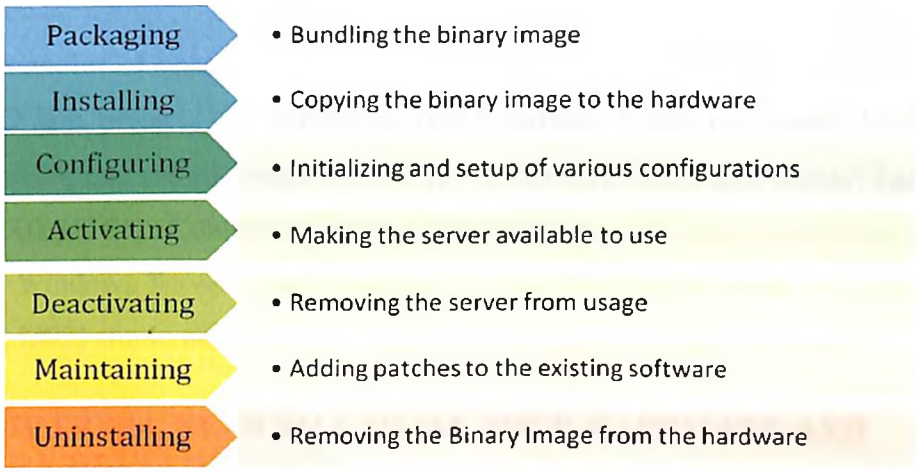


Figure 2-17 Activities for an application binary in the enterprise

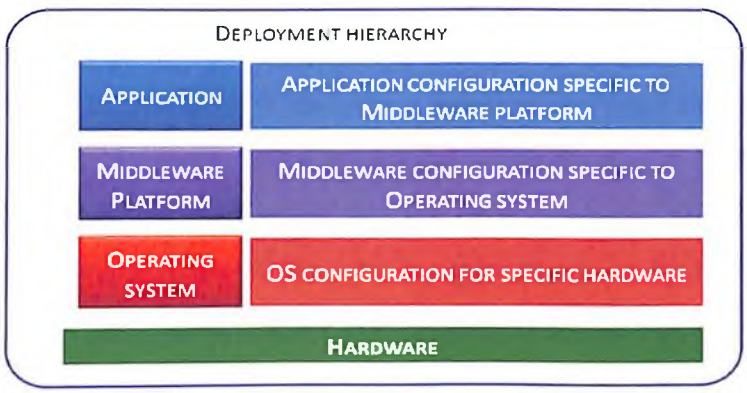


Figure 2-18 deployment hierarchy

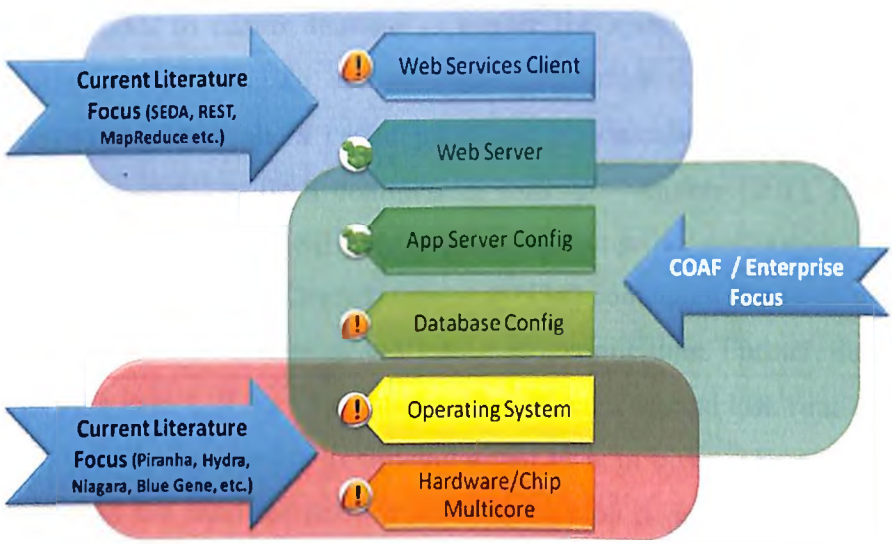


Figure 2-19 Focus area of the literature and COAF

Important examples of operating system level deployments specifically targeted to Multi-core are FreeBSD Jails project [131] for simple UNIX partition, Linux containers [151], Solaris containers, AIX workload partitioning (WPAR) [240], Parallels [185] and MontaVista Carrier Grade Edition (CGE) [164]. Commercial implementations such as Tivoli Provisioning Manager [118], Microsoft Windows Server Update Services [[218]], Red Hat Network, etc., and custom prototypes [237], [207] aim to automate the deployment process across all the layers.

2.4 ARCHITECTURAL NEED TO UTILISE BOTH HARDWARE AND SOFTWARE DEVELOPMENTS

2.4.1 Enterprise studies and the need for new system architectures

Enterprise class software platform manufacturers like IBM, Oracle etc., and hardware platform manufacturers like Intel, Dell, HP etc., have exposed the need for new application architectures to accommodate Multi-core in the enterprise. Knauerhase et al [136] observed the dynamic run time behaviour of the application and the memory cache; they highlighted the inadequacy of existing operating systems to handle the complexities arise due to sharing of cache in the context of Multi-core processors. Khun Ban et al [23] devised performance analysis methodology for a Java based application to demonstrate the need for a new architecture that leverages hardware performance improvements of Multi-core. Kim et al. [135] have identified the need to explore methods to ensure fairness to ensure the system level parameter (cache contention) is uniform across tasks. To accommodate performance at the user space, they proposed a dynamic scheme that alters cache partitions periodically from the immediately previous repartitioning. Researchers from commercial hardware vendors [201], [3], [86], and [87] have sought a new solution that will help monitor and improve performance of generic enterprise applications on Multi-core. Knauerhause et al [136] contended the near impossibility of determining the behaviour of an enterprise application at compile time. Further, they observed that when a task runs on a core 1, it uses the entire cache. When a second task runs on core 2, it shares the cache with core 1, thus results in slowing down of both tasks. The quantification of slowdown is dynamic, and the resultant performance degradation depends on individual task's behaviour at a given time, thereby highlighting the temporal characteristics of the enterprise application. Altman et al. from IBM [5], highlighted the lack of intelligent thread migration

methodologies for scheduling the workloads amongst multiple cores, thus cite the impact of spatial characteristics of the enterprise application. Intel Corp [23] identified the need for architecture specific information at the application level to optimally schedule the workload specific to an asymmetric features such as die area, etc. of Multi-core. Barroso et al. [26] corroborated the need for examining the hardware level execution characteristics and the application level execution characteristics for optimally scheduling enterprise workloads. Their analysis considers both user level and kernel level execution statistics and highlight the need for request-level behaviour characterization.

2.4.2 Summary of the Review of Literature

This chapter reviewed the state of the art in hardware and software architectures in the areas of Multi-core servers and web services based applications respectively. It also discusses modern methodologies that tackle Multi-core, memory and software architecture advances individually. Furthermore, this chapter determines the need to have an architectural approach that marries the best practice available in these architectures.

First we identified the general properties of the enterprise application, namely spatio-temporal properties and stateful properties. Next, we studied and analysed number of existing architectures, and solutions that have been proposed and prototyped for exploiting parallelism in both hardware and software. There are definite advantages and best practices that we can directly borrow from these architectures. The first step is to make this business logic to be stateless. To achieve this, the application needs to modified or rewritten or reconfigured. Modifying a well-running application for the sake of incorporating a new hardware such as Multi-core may not be a pragmatic approach in the enterprise. Therefore, we need a generic external mechanism to identify these bottlenecks. These bottlenecks need to be correlated to the associated business logic and the computing resources need to be relocated for better or optimum utilization.

Figure 2-19 highlights the focus for this thesis on improving the performance of the enterprise applications, by observing at the operating system level and accordingly tune the configuration parameters using the best practices and architectural approaches that are highlighted in this literature.

3 DESIGN OF A CONNECTION-ORIENTED ARCHITECTURE FRAMEWORK

In the summary of the chapter two, the need for a new architecture framework that overlays and delivers the power of Multi-core hardware is envisioned. This architecture framework enables web services based enterprise applications use the connected context and thereby provides a rich user experience. This new architecture framework called, Connection-Oriented Architecture Framework on Multi-core (COAF) [233] for enterprise class web services, is proposed. This new architecture targets the next-generation of enterprise applications based on web services. Enterprise applications have a specific behaviour and characteristics, and are usually averse to any form of change, which is perceived as a risk, especially at the source code level. COAF is built to leverage the compute capabilities of the Multi-core without modifying the existing web services based applications.

COAF introduces two key concepts namely "Connection-Orientation" and "Infrastructure Awareness" and these two concepts are abstracted in six modules. COAF can be used to develop and deploy enterprise applications of all kinds, which include general purpose applications like Microsoft outlook, ERP systems such as SAP, Oracle financials, or custom made applications such as core-banking system, ware-house management system, credit card processing systems etc. and generally designed to be multi-threaded. A COAF will help leverage multiple threads that are inherently available in Multi-core based hardware server platforms to serve the multi-threaded software applications. COAF also helps the connectionless web service based enterprise systems to work like Connection-Oriented stateful system, thereby provides a rich user interaction patterns.

3.1 COAF ARCHITECTURAL CONSIDERATIONS

Based on the previous studies and observations in the literature survey, the following considerations are summarized for designing the COAF architecture. Enterprise literature is full of knowledge of existing best practices

Web services are given the treatment of first class citizens of the next generation enterprise architecture. Web services abstract both software implementation of business applications such as CRM application, ERP application, Human resources application etc., and hardware devices

such as Multi-core platform, flash memory etc. This "First class citizen" treatment for these web services implies that we would be able to construct, operate and manage these services at run time. Literature indicates that web-services will be largely used to construct business applications that can be hosted on cloud in "Software as Services Model". Enterprise research and current trends indicated in Section 2.4.1, elicit the following key architectural considerations for hosting enterprise class web services on Multi-core infrastructure.

1. Leverage matured multi-threaded application technologies and frameworks and evolving technologies such as cloud. This is based on the assumption that enterprise deployments are heterogeneous and enterprises prefer to have their own private clouds [52] [243] in the future. This means that the architecture needs to be inclusive in nature to plug various components of the systems, such as operating systems (various Linux distros, various flavours of Unix like Solaris, Windows etc.), databases such as MySQL, Oracle, SQL Server, Teradata etc., application server environments like Websphere, Weblogic, JBoss, and application frameworks such as Spring, Hibernate, Struts, .NET etc. This consideration assumes that the underlying technologies for these components are natively multi-threaded.

2. Be agnostic to hardware virtualization techniques such as Linux OpenVZ [170], VMWare [238], Xen hypervisor [251], Solaris Zones and Containers etc.

3. Leverages run-time deployment mechanisms that exist in various deployment environments such as the deployment descriptor framework JSR-88 for Java [199], Mina network application framework [14], and .Net deployment toolkits [205] for assembly. This means the architecture needs to be pluggable.

4. Has a mechanism to abstract system management technologies and tools available in open source and commercial licenses, such as Apache [10], HP OpenView [114], IBM Tivoli framework [118], Oracle management framework [175], etc.

5. Abstract different instrumentation mechanisms technologies such as Intel vTune [125], LTTng [148], Solaris DTrace [156] etc, so that underlying system can be

monitored and analysed for the system performance. Comparison table for probes in mentioned in Appendix I.

6. Strive to provide the aspects of Connection-Orientation till the execution end-point for the web services. Emphasize for Connection-Orientation in COAF for enterprise systems comes from the underlying fact that the enterprise class systems require relatively "long-lived sessions", and require very rich user interfaces, when compared to current-generation web-service designs and architectures such as Yahoo, Google etc. which are "short lived sessions". Web servers such as Apache and web application execution environments such as J2EE, .NET, typically provide native support for application level session support for use-cases like shopping-cart style. This state may be managed either on in-memory or on external server, for example in J2EE environment it is typically HttpSession object used for maintaining the session.

7. Support configuration management. This is required for the architecture, so that changes can be made to the existing configuration, instead of resetting the entire system.

8. Support hot deployments, so that the scheduling policies can be enforced in real time on the working system without stopping the system. This requirement is also enforced due to the availability requirements [41] of the business applications.

9. Support the ability to receive and send real-time and near real-time notifications. This is important for the system to receive and send the real-time feedback about happenings in the system, so that scheduling policies can be adjusted to adapt to those changes.

10. Complement existing operating systems with additional information instead of major upgrade or change to existing operation environments.

3.2 KEY ARCHITECTURE CONCEPTS

The core architectural considerations listed in previous section are abstracted and realised using two key architectural concepts. They are:

1. Scheduling for "Connection-Orientation" using contextual service dispatch mechanism
2. Deployment configuration using "Infrastructure Awareness" using threshold values obtained for various configurations using empirical data and feedback using metrics such as "Cache miss".

3.2.1 Scheduling for Connection-Orientation

In an application, typically the request flows from user space (application) to kernel space (where the operating system schedules the task) to hardware as depicted in Figure 3-1. During this process of request transition from user space to kernel space, context specific to application (user request) is partially lost. This context loss is prevalent in the web services based systems, where the web-service arrivals are partially non-deterministic.

The context loss makes it difficult for operating systems to make the correct assumptions about caching and scheduling the process. This loss is highlighted and distinguished as differences between the Figure 3-2 and Figure 3-3. TCP ensures the reliability of connection and guarantees protocol connection between two network endpoints. However, after reaching the server side endpoint the connection context is lost, as the request enters the service side infrastructure for processing. Once, the response is created on the server side, TCP takes the control on connection. Typical processing model for web services on the server side is stateless, so an extra effort is required to store the connection context. This additional information extends the connection upto the core compute endpoint instead of network endpoint. Thus, Connection-Orientation enables the scheduling of the application process in the right core, so that scheduling and processing efficiency of the application process is improved.

Figure 3-1Transition of Request from User to Hardware

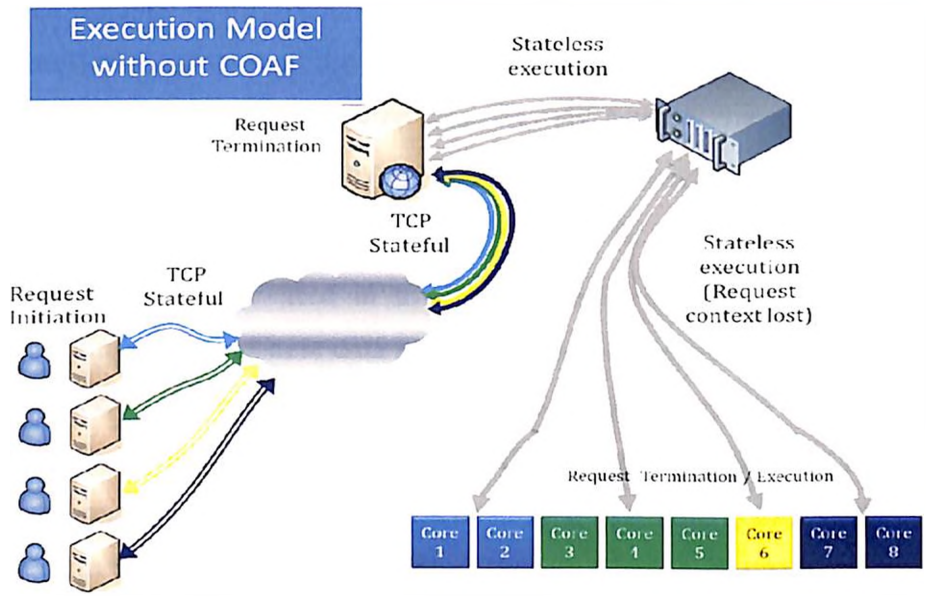


Figure 3-2 Execution under current connectionless architecture

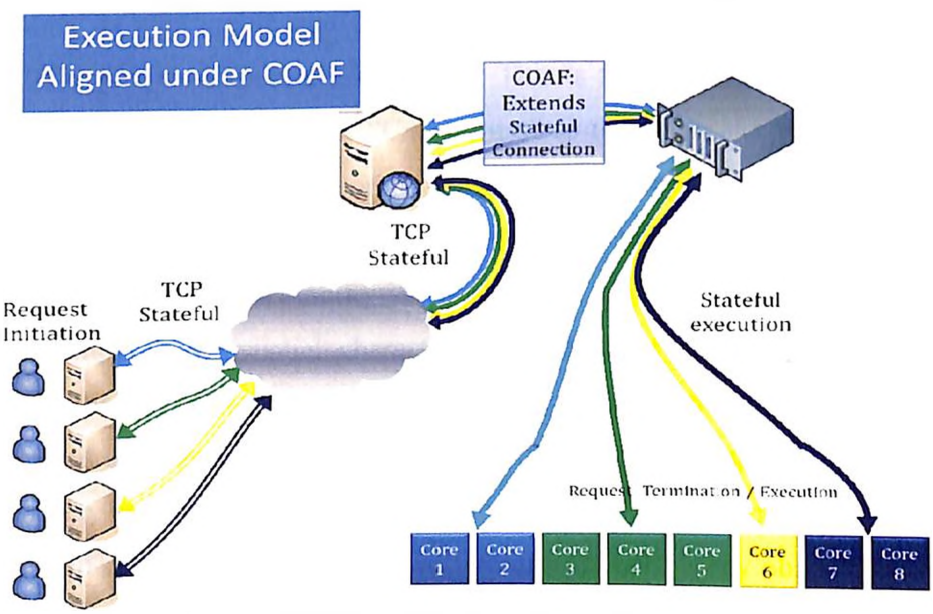


Figure 3-3 Need for improvement using Connection-Oriented Architecture

3.2.1.1 Cache miss is the key metric

Once the system starts to execute user space requests such as web service requests, the request is ultimately converted into data and instructions sets for the kernel space. For example in the Figure 3-4, clients (client 1, client 2, client 3, ... client n) are requesting an account summary for their accounts in the financial enterprise. There are different priorities assigned based on the type of the account. Margin account is preferred to retirement account. The account summary template (and the associated the rendering logic / the instruction at the browser) is common across all these service requests. As listed in Table 3-1, the record size that is fetched for each account type may vary. All the above application level contexts are lost when a request goes from application to operating system.

The client request gets converted in to data and instruction sets, while moving from application to operating system. These data and instructions sets move towards processing core, from disk, to memory, to cache, to register. During this movement, if the cache did not contain data relevant to operating instructions, then Cache miss happens, and the data is again fetched. When a process encounters more number of Cache misses, operating system pushes the current application process to lower priority compared to a process which performs with less number of Cache misses. This results in a "bad user experience". Fedorova et al demonstrated [77] that "Cache miss count" is measurable and predictable, and contention for the L2 cache has the greatest effect [255] [36] on system performance. This is further corroborated with Hartstein [110], who elucidated the importance of "Cache miss" as the parameter that needs to be used and understood. They begin with the theoretical limit proposed by Chow [50]. Their reasoning for "Cache miss count" at operating system level can be correlated to a configuration parameter at the application level thereby facilitating the traverse of application context to the operating system for both temporal and spatial characteristics. Tseng et al. [224] identify the multi-threaded nature of many commercial applications makes them seemingly a good fit with the increasing number of available Multi-core architectures for enterprise applications like SAP-SD and IBM Trade. They evaluate the performance scalability and the thread-placement sensitivity, increasing number of cores increasing number of threads per core. They observe that enterprise applications hide long latency memory operations (i.e. L2 misses) in a Multi-core system.

Table 3-1 Table showing the number records and time to fetch for each account type

Account Type	Records	Size (KB)
Individual	2305	117
Global trading	2192	112
Retirement	2250	115
Margin	2114	110

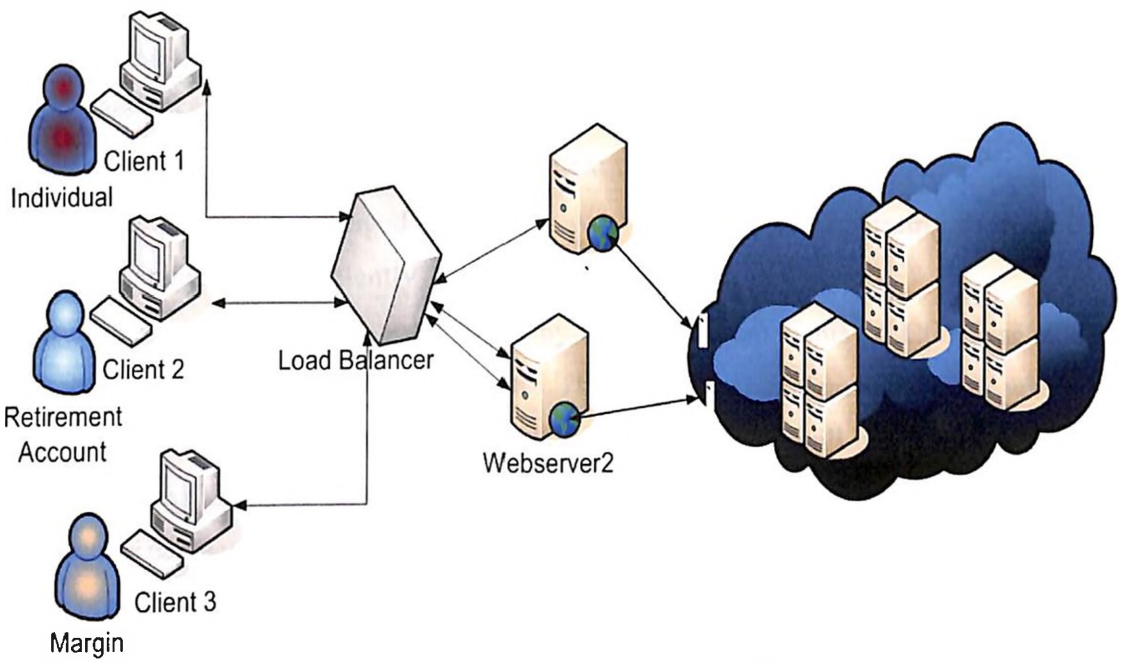


Figure 3-4 Example scenarios for clients requesting Account summary for various accounts

The operating system is application agnostic, and hence, it cannot recover its performance (reduction of Cache misses) without the knowledge of the application context. The performance enhancement is limited by the dependence of the Cache miss rate on cache size the square root rule as determined by Chow [50] in Equation (1)

$$M = M_0 C^{-p} \qquad (1)$$

where M is the Cache miss, C is the cache size and the exponent, p, typically takes on values between 0.3 and 0.7. The rate of re-referencing for a specific cache line is also theoretically determined from Hartstein [110], as in Equation (2)

$$R = R_0 t^{-\beta} \qquad (2)$$

where R is rate of re-references which is a decaying function of time t. The decay constant, β, is found to take on values ranging from 1.3 to 1.7. Due to this reordering of the priority at the operating system and the latency due to re-execution of instructions, the end user gets a disconnected experience. The user experience will continue to get richer if

- there is a way to pass the application context to operating system or
- these requests are arranged in such a way such that the operating system uses processing context efficiently

The two above requirements are established by two key components of COAF scheduler module namely, ServiceMap and SchedulerState. The COAF scheduler will know where to dispatch the request using the ServiceMap component; the scheduler will also know the count and status of number of requests that are currently processed, using SchedulerState component. Thus, the scheduler can schedule the request to existing server or a new server. This contextual information about the service scheduled ensures that the subsequent request can be dispatched to appropriate core. This leverages spatio-temporal characteristics, thus ensuring the performance local aspects to process the request. Therefore, in context, the key system level parameter that correlates to the application performance is "Cache miss count".

3.2.1.2 Contextual service dispatch

Application context is available for affecting the behaviour of the application. Application context can be spatio-temporally local, when the request dispatch is synchronized with instructions and data. Two kinds of contexts namely design time context and run time context, which together encapsulate the locality characteristics of the application. (a) Design time context

constitutes the prior knowledge available at the time of design with the developer about the complexity of the service implementation, stateful or stateless nature of the service request, IO or compute intensive nature of the service, read-mostly or read-write or write-only nature of the service etc; deployment environment information includes light weight Apache, which MySQL storage engine, which version of Tomcat etc.; server capacity configurations such as dual core, memory size, memory classification such as SSD, flash, DRAM etc., cache size etc.; configuration parameters for the underlying system such as thread pools, timeouts, thread priorities etc. (b) Run time context constitutes the knowledge gained in production environment of the application about amount of memory is being consumed, number of system processes that are involved in processing the current service request, number of Cache misses that have happened etc. Both the design time and run time contexts are needed to contextually dispatch the service to the appropriate server.

3.2.2 Deployment configuration using Infrastructure Awareness

Enterprise applications are deployed on a heterogeneous infrastructure; for example, the same application can be deployed on a single core server on a 2GB main memory or deployed on a 32 cores server with 16GB main memory etc. Similarly, the infrastructure itself can have various configurations; for example, a same 32 cores server with 16GB main memory can be configured to accept 200 simultaneous threads or 1000 simultaneous threads. This knowledge about the infrastructure and the deployment configuration is available to the enterprise administrator. This knowledge is termed as "Infrastructure Awareness" and it can be used to configure the system for optimal utilization. Parameters of the Infrastructure Awareness for Multi-core that affect the performance of the system include, number of cores, interconnect latency, main memory, size and organization of L1 and L2 memory cache, memory fetch latency, cache coherency models such as "In-order" or "out-of-order", 32 bit or 64 bit, availability of special processing capabilities such as XML streaming etc. Same service request will have different processing behaviour based on the combinations of the above parameters. Hence, it is important to understand these constraints to model the configuration of the overall system. Once modelled, these configurations are used to arrive at threshold values. These threshold values are "infrastructure aware" deployment configurations.

3.2.3 Performance model of COAF

In addition to "Cache miss count", "contextual service dispatch" and "Infrastructure Awareness", two kinds of control mechanisms are used to tune the system continuously for improving the utilization of the Multi-core server. They are admission control and feedback control. Admission control is about controlling the number of requests that are dispatched to the server, which is based on the application knowledge available at the time of deployment of the configurations such as deployment descriptors, base line threshold configurations. Feedback control is achieved by observing the run-time behaviour of the system against the pre-computed threshold values of the deployment parameters. The feedback itself is received from the run time logs of system events that are correlated to the application service patterns. To achieve these two control mechanisms, two parameters namely "Core Configuration Effectiveness" and "Intra Process Efficiency" are derived and these two parameters are detailed in Section 5.3.1 and Section 5.3.2.

3.3 COAF – AN AMALGAMATION OF HARDWARE AND SOFTWARE

Figure 3-5 represents the marriage of innovations in hardware architectures such as Multi-core with innovations in software architectures such as web services based service oriented applications using COAF. This amalgamation has three major contributions; (1) establishes Connection-Orientation design for the enterprise web services using additional state information by explicitly marking the temporal and spatial locality characteristics of the web service; (2) leverages the knowledge of the infrastructure to adjust the threshold of the system configuration; and (3) ability to model both software and hardware resources as web services so that the application architecture can be configured with various implementations during deployment.

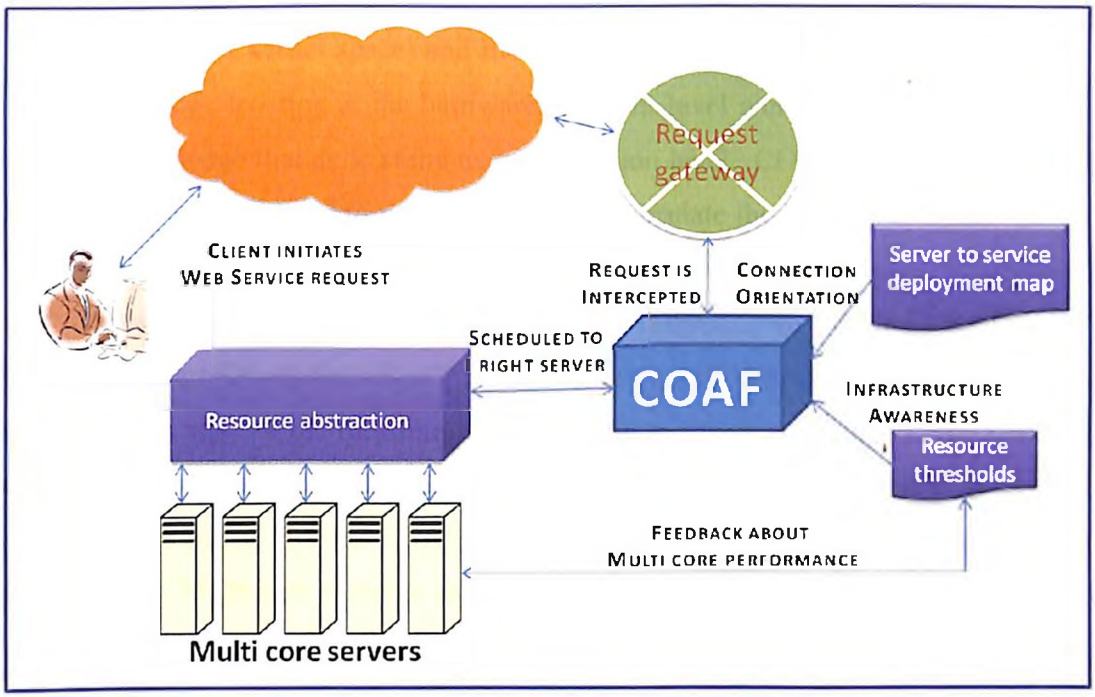


Figure 3-5 COAF – An amalgamation of Multi-core with enterprise web services

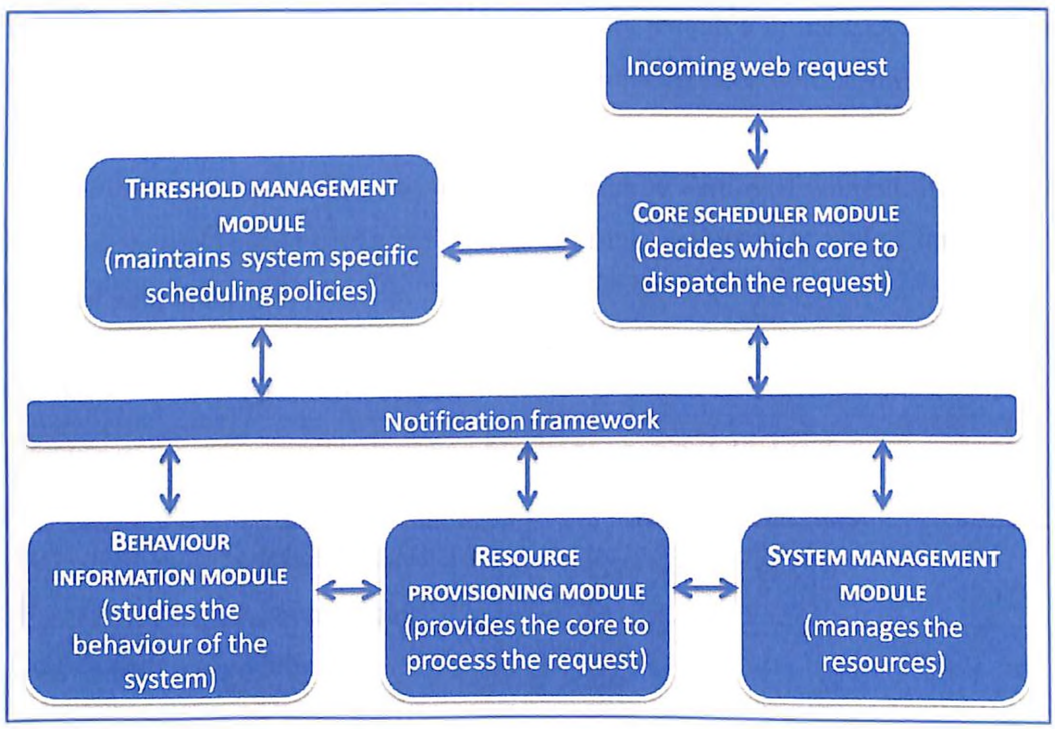


Figure 3-6 Six modules of COAF and their interfaces

The core focus of COAF is to exploit the compute capabilities provided by recent and next generation Multi-core platform, and it approaches the problem from both top-down (at user space) and bottom-up (at kernel space) and thereby creates a hybrid solution that leverages the fine-grained knowledge existing at the hardware platform level namely CPU, IO, Memory and coarse-grained knowledge that exist at the user-application level. COAF houses instrumentation mechanisms at fine-grained level to understand and to correlate the behaviour of the Multi-core platform to its configurations. The combined knowledge of the system at design time and at run time using feedback systems is used to setup and tune the optimum threshold for the system, such that the multiple cores or threads at the server are engaged to provide optimum throughput. Based on the current state of the run-time, the admission of future requests is controlled and the web service flow is managed. The modelling of both hardware and software as services is holistic in approach and it enables COAF to provide pluggable architecture that can leverage future evolutions in Multi-core systems and large memory technologies. COAF uses enabling technologies such large memory RAMs, solid state disks, flash based RAM to support its Connection-Orientation paradigm.

COAF is scalable due to its implementation agnostic approach to its underlying services, for both hardware and software services. The modelling of any resource in the COAF system as a web service provides an opportunity to plug and play various services both at the infrastructure level and application level, thereby ubiquity of COAF is established. Components of the COAF themselves are web-services, and hence the architecture can be deployed without the overheads of installing any new environment. COAF has a modular architecture which improves the maintainability and simplicity.

3.4 MAIN COMPONENTS OF COAF

COAF consists of the following six building blocks as shown in Figure 3-6.

1. Resource provisioning module
2. Behaviour Information module
3. Threshold Configuration module
4. Core Scheduler module
5. System Management module
6. Notification Framework module

Resource provisioning module provisions a resource for processing the client service requests. Resource is modelled as web service, and the resource could be a software resource or hardware resource. System Management module deploys and undeploys the resource, and manages the lifetime of the resource. Behaviour Information module provides the run-time behaviour information of the resource using the non-invasive instrumentation mechanisms. This knowledge obtained from Behaviour Information module is used as feedback for computing new thresholds or confirming the existing threshold for optimum performance of processing servers. Notification Framework module enables the communication between modules. Threshold Configuration module houses threshold values for configuring each resource that is deployed in the system. Core Scheduler module dispatches the web service request to appropriate server based on the ServiceMap and SchedulerState information thereby provides connected oriented experience. These responsibilities are illustrated in Figure 3-6.

3.4.1 Resource provisioning module

Resource provisioning module provides the resources for processing any web service request. Two key components of this module are resource and the Resource Manager. As depicted in the diagram below, the resource is a web service, and this web service could be either software or hardware. The choice of modelling a resource as web service enables a standard interaction pattern in various ways without affecting the existing setup. This also helps to modify / enhance the resource implementation technology without affecting the architecture.

3.4.1.1 COAF Resource is high level of abstraction:

Resource is the highest level abstraction of all resources that are deployed under COAF architecture. The Resource could be a software or hardware. COAF treats the Resource as the first class representation for its architectural framework. COAF exposes any resource in the framework as a web service.

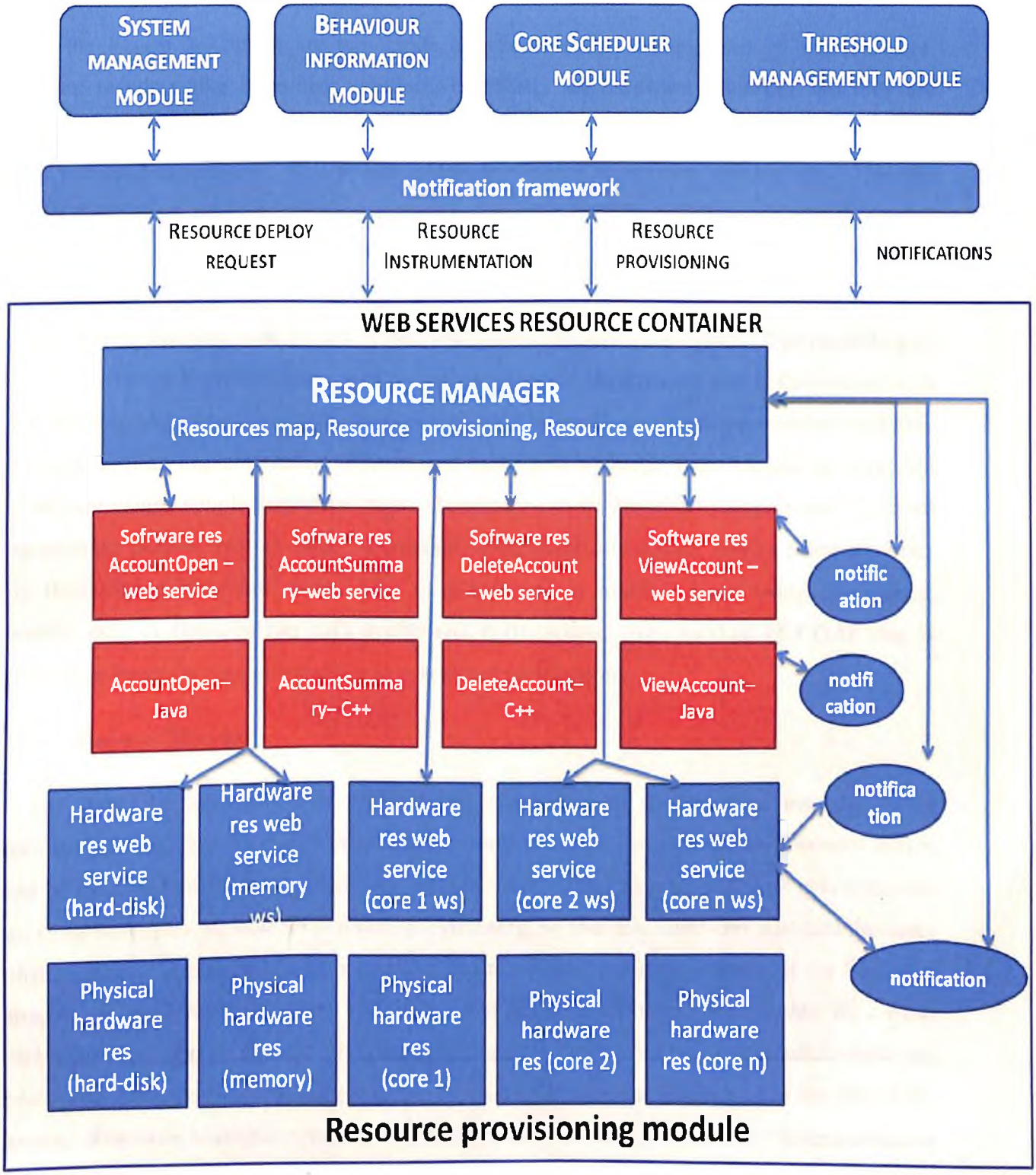


Figure 3-7 COAF Resource Provisioning Module

In the Figure 3-7, there are two kinds of resources that are exposed as web services. Hardware resource like Hard-disk, memory, core1 etc, are hardware resources and they are exposed as web services to the consumers. Similarly, software resources like OpenAccount, CreateAccountSummary, etc. are software resources exposed as software web services. This one level of abstraction helps to plug any appropriate implementation into the architecture, so that COAF can be complemented with both existing architectures and new architectures. This abstraction as a web service also leverages the virtualized architectures.

A resource interacts with its eco-system using this web service interface. The modelling of web service based interface makes web as the platform for deployment and collaboration with other services. Any implementation can be converted into web service using standard tool kits. For example, if an implementation is developed using Java language, then it could be converted into service using simple annotation tags. Resources can be modelled using various standard programming models such as object orientation, component orientation, service orientation etc. This flexibility enables the composition of a resource in multiple ways using inheritance, assembly etc. A resource can emit events and notifications. Any module of COAF that is interested in these events can subscribe to associated notifications.

3.4.1.2 Resource Manager

The Resource Manager is an independent component of the resource module. This separation of Resource Manager helps the Resource Manager to manage the resource which could be local or remote and can live any process space. The Resource Manager gets a request from Core Scheduler module for resource provisioning, so that scheduler can schedule the tasks to that resource. Resource Manager provides the resource if it already exists. If the Resource Manager does not have the resource then it requests the System Management module for a fresh deployment of a resource through a notification. Once the System Management module deployed the resource, Resource Manager can start to provision the resource, till the end of the life of the resource. Resource Manager interacts with Behaviour Information module for instrumentation of a resource, and with Threshold Configuration module for providing notifications and events about the resource.

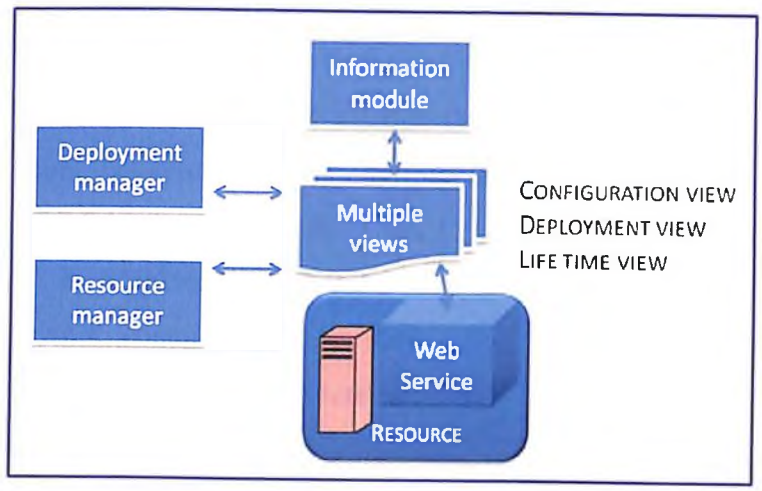


Figure 3-8 Provisioning a Resource

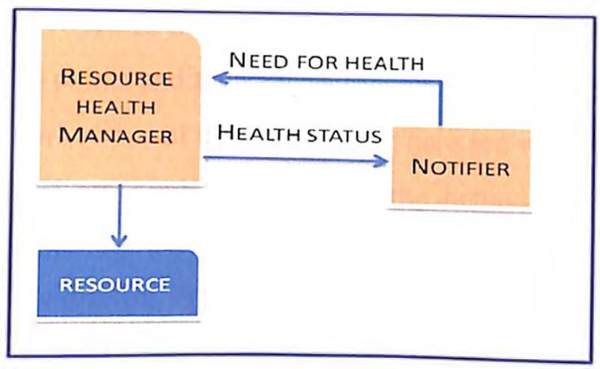


Figure 3-9 Resource Manager

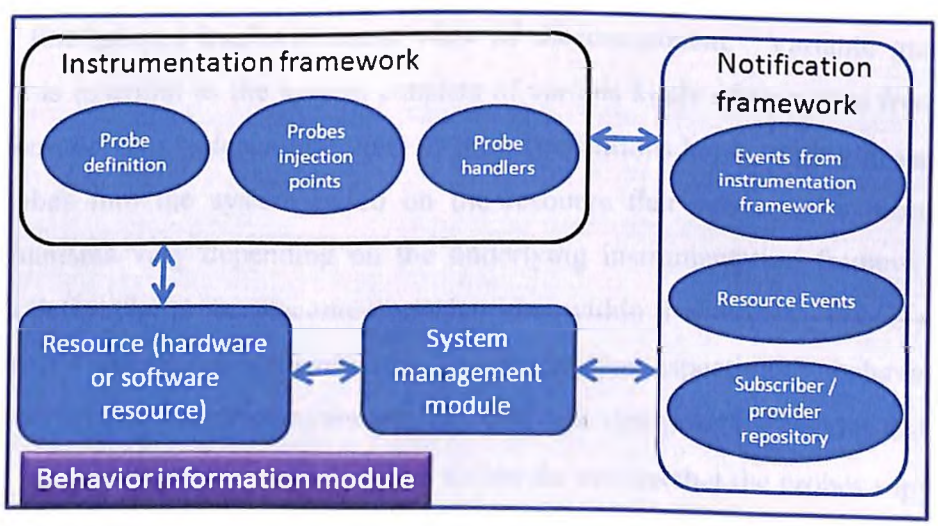


Figure 3-10 Behaviour Information module

3.4.2 Behaviour Information module

Behaviour Information module provides the run-time behaviour information of the system. It deals with two kinds of state changes that happen within an enterprise infrastructure. First is about the status of the resource itself – like whether the resource is live, running, waiting, etc. and the second is about behaviour of the environment. Both these information together determine the behaviour of the overall system. Thus, COAF provides a comprehensive approach to manage the system in a closed loop. Behaviour Information module consists of Probe Instrumentation framework containing probe definitions, probe injection points and probe handlers, and the notification agent that interacts with System Management module and Resource Provisioning module.

3.4.2.1 Probe Instrumentation framework:

Probe instrumentation framework as depicted in Figure 3-11 consists of various probes and their definitions, probe injection points and their respective handlers. Probes are instrumented at the resource level to know the behaviour of the resource. There are various kinds of probes and the associated frameworks [223] are used for enquiring the behaviour of the resource. The probing could be static or run-time, intrusive or non-intrusive. There are three components that constitute the instrumentation framework. They are probe definition, probe injection points, and probe handlers. Probe definition defines the granularity of the probing. Probes could be macro level probes which provide a highly abstracted view of the component or micro level probes which provide fine-grained implementation view of the component. Variable granularity of probe definition is essential as the system consists of various kinds of resources from hardware to software resource. This independent view of probe definitions helps to plug in various kinds of pre-built probes into the system based on the resource that needs to be probed. Probe definition mechanisms vary depending on the underlying instrumentation framework. Probe injection points define the probe placement mechanisms within the system. This is essential to isolate and inspect a specific component within the system like inspecting the behaviour at main memory level etc. Instrumentation framework picks the injection point definitions and places the probes in those injection points. Probe handlers define the actions that the probes supposed to do to get the measurements. The actions could be streaming of events to a specific notification system, or writing the log to a file, or calling back some other action within the framework or

any provider within the system. The information gathered from probes is used in various ways, like setting up a baseline and threshold values, mapping the application needs to the underlying infrastructure, and fine-tuning the thresholds by correlating the notifications that are emitted from the web-services to the application level events. These information providers are both software and hardware probes that can be instrumented at various levels to provide the information across various layers of the service, from application layer to operating system layer to hardware level. The probes could be both non-intrusive and intrusive and they could be abstracted as a web service, so it can be deployed within the web services framework. The information builders are implemented as web services, so that the interface for interacting with other web services are standards based and do not require any special execution environment for plugging them. These information builders can be setup to operate and capture the run-time information at various levels.

3.4.3 System Management module

System Management module implements all the functions required to deploy, to undeploy and to manage a resource in COAF framework. System Management module works in tandem with Core Scheduler module, Resource Provisioning module and Threshold Configuration module. The internals of the System Management module is given in the Figure 3-12. System Management module contains two components: (1) Deployment manager and (2) Notifier and the design of these components as follows:

3.4.3.1 Deployment Manager:

Deployment manager is the key component of System Management module. System Management module receives the request for deploying a resource from the Core Scheduler module. Deployment manager picks up this request for deploying the service. Similarly, System Management module receives the message for undeploying a resource, which is also passed to deployment manager. Thus, deployment manager does two things: (a) Deploy the resource and (b) Undeploy the resource. The deployment manager interfaces with configuration management module to fetch the configuration information required for deploying the resource. Figure 3-13 depicts the components and the workflow in deploying a service.

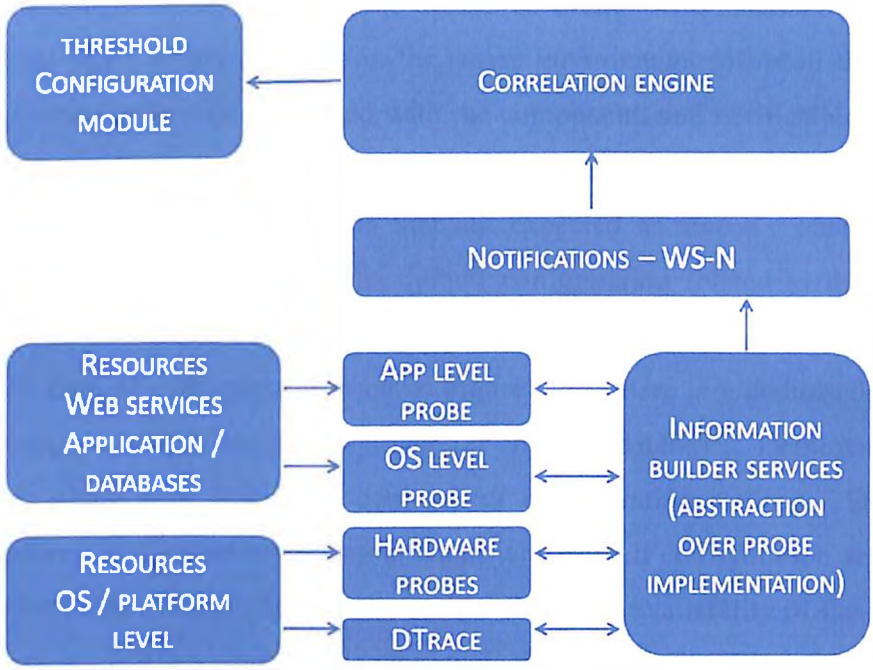


Figure 3-11 Probe Instrumentation Framework

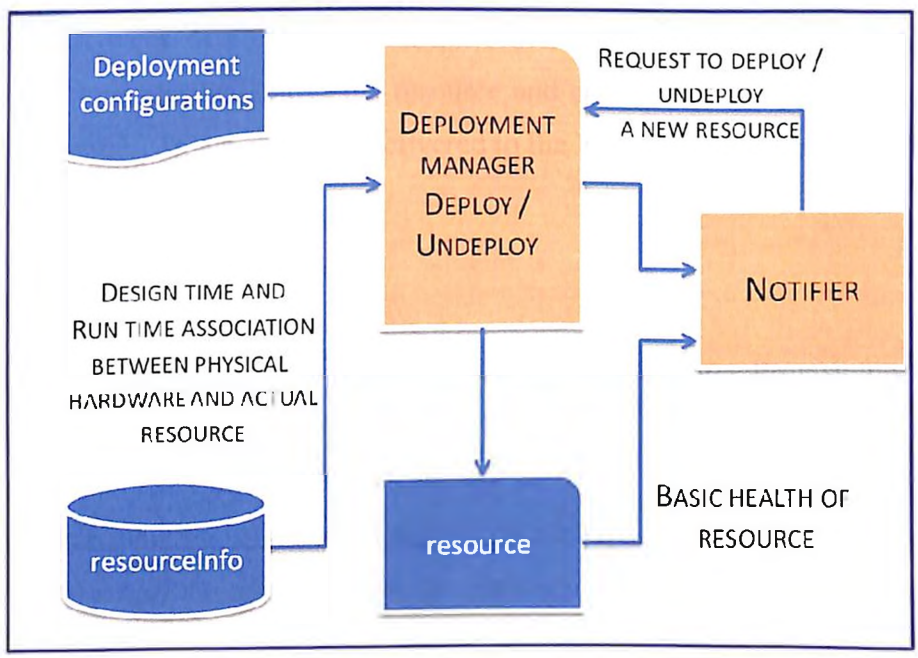


Figure 3-12 Three Components of the System Management module

Deployment Manager subscribes to deployResource event. This message consists of the information such as name of the component, the server environment to install the component, domain to bind, configuration file associated with the component, and notification message that need to be sent in case of success / failure. Once the deployment manager gets the notification, it dissects the message to identify the details that are expected as above. Then the manager requests the configuration server to pull the default configuration related to that component. This configuration server is maintained by Threshold Configuration module. There is a deployer dedicated for each kind of component as seen in Figure 3-12; there is a dedicated deployer for Apache web server; similarly there is a deployer for MySQL database. Deployment manager calls the deployer along with associated deployment configuration settings. Deployer then installs the component, and initializes the component with default configuration settings. Once deployed, the deployment manager creates a message about the availability of the resource and hands the message to Notifier. Similarly, Deployment manager gets the undeployResource message from the Notification Framework module. Request for undeployment of a resource can come for multiple reasons; for example, a resource may go into non-steady state and become unavailable or system does not require the services of a resource (process_payroll service is needed only in the last week of the month and not required for the remaining periods of the month). Deployment manager undeploys the resource and creates the message about the non-availability of the resource. This message is delivered to the Notification Framework module by the Notifier.

3.4.3.2 Notifier

Notifier subscribes to messages from various publishers like Resource Provisioning Module, and Core Scheduler module through Notification Framework module. Similarly, Notifier publishes the messages to Core Scheduler module and Resource Provisioning module through Notification Framework module. Notifier subscribes to message from Core Scheduler module for deployResource message. Notifier receives the message deployResource from Notification Framework module, and splits the message header into various parts that are important for deploying the resource. Then, it passes this information to deployment manager. Similarly, Notifier receives the message for changing the configuration parameter value for a resource. Notifier then dismantles the message and passes relevant information to deployment manager for

changing the configuration parameter of the resource. Once the resource is deployed in to the system or undeployed from the system, Notifier publishes the messages appropriately to the Notification Framework module.

3.4.3.3 Configurations

Configurations are the deployment instructions that are required for deploying the resource. System Management module uses these configuration files to setup and initialize the resource while deploying it. Configuration files are in text file format. These files are stored in the Configuration Server that is maintained by the Threshold Configuration module.

3.4.4 Notification Framework module

Notification Framework connects all the other modules of COAF. It transports the messages about the events that happen in the system between the COAF modules. System events are converted into notifications. These notifications are submitted to Notification Framework module for transportation. There are three components of the Notification Framework module, namely Subscription manager, Event consumers and Event producer as shown in Figure 3-14. Subscription manager maintains the information about validity of the subscriptions of event consumers; it also provides the status of the subscriptions for event consumers. Event consumer has to subscribe for events; the subscription is valid for a particular time period. If the subscription has expired, then event consumer can renew the subscription or unsubscribe for receiving the event notifications. There are various notifications corresponding to the events happen in the system. System Management module can publish the health of the resource; Resources themselves can publish the notification about their health; Behaviour Information module provides notifications about start of the information collection, and about the availability of collected information for further correlation; Threshold Configuration module sends notifications about changes in the deployment configuration parameters; Core Scheduler module can request for the deployment of a new resource; Notifications framework is the important module in the COAF, because the other modules are dependent on the notifications provided by the framework for further actions. Notification Framework module maintains the repository of providers and subscribers and ensures the notifications are appropriately dispatched to the subscribers, and ensures [181] all the messages are delivered during the dispatch process.

changing the configuration parameter of the resource. Once the resource is deployed in to the system or undeployed from the system, Notifier publishes the messages appropriately to the Notification Framework module.

3.4.3.3 Configurations

Configurations are the deployment instructions that are required for deploying the resource. System Management module uses these configuration files to setup and initialize the resource while deploying it. Configuration files are in text file format. These files are stored in the Configuration Server that is maintained by the Threshold Configuration module.

3.4.4 Notification Framework module

Notification Framework connects all the other modules of COAF. It transports the messages about the events that happen in the system between the COAF modules. System events are converted into notifications. These notifications are submitted to Notification Framework module for transportation. There are three components of the Notification Framework module, namely Subscription manager, Event consumers and Event producer as shown in Figure 3-14. Subscription manager maintains the information about validity of the subscriptions of event consumers; it also provides the status of the subscriptions for event consumers. Event consumer has to subscribe for events; the subscription is valid for a particular time period. If the subscription has expired, then event consumer can renew the subscription or unsubscribe for receiving the event notifications. There are various notifications corresponding to the events happen in the system. System Management module can publish the health of the resource; Resources themselves can publish the notification about their health; Behaviour Information module provides notifications about start of the information collection, and about the availability of collected information for further correlation; Threshold Configuration module sends notifications about changes in the deployment configuration parameters; Core Scheduler module can request for the deployment of a new resource; Notifications framework is the important module in the COAF, because the other modules are dependent on the notifications provided by the framework for further actions. Notification Framework module maintains the repository of providers and subscribers and ensures the notifications are appropriately dispatched to the subscribers, and ensures [181] all the messages are delivered during the dispatch process.

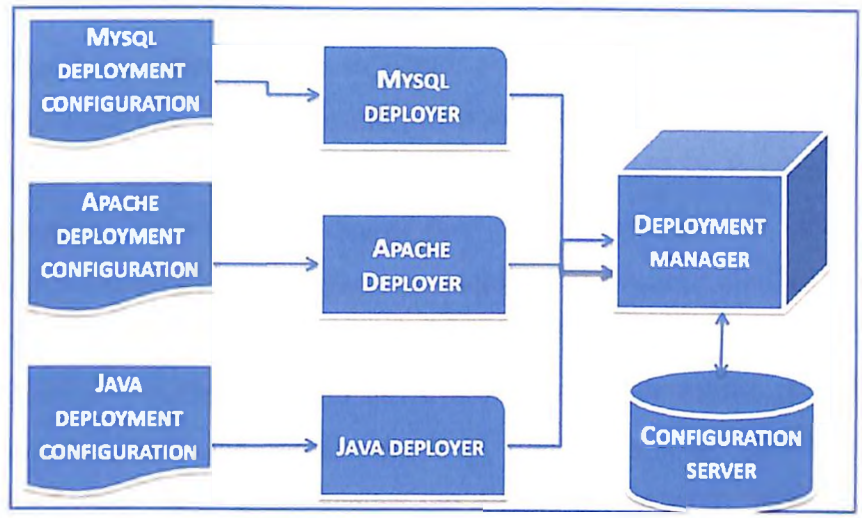


Figure 3-13 Deploying a Service

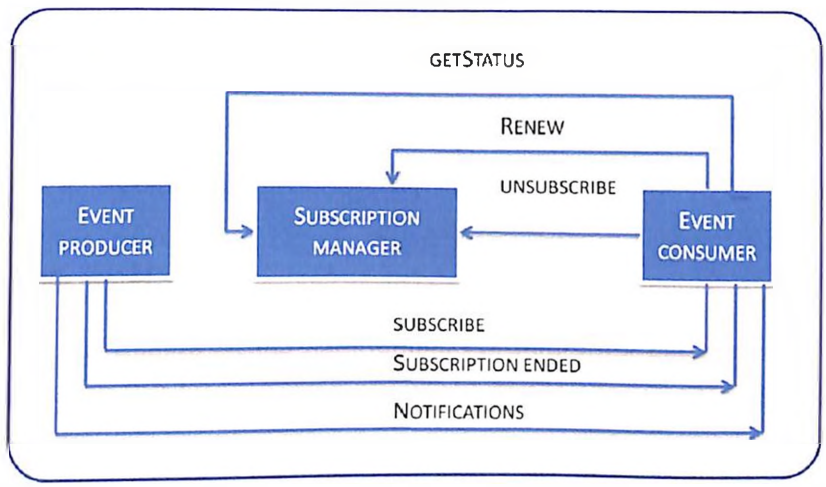


Figure 3-14 Notification Framework module

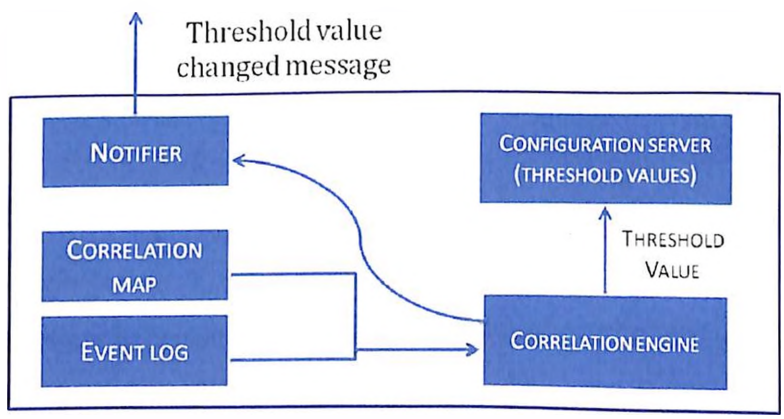


Figure 3-15 Threshold Configuration module

3.4.5 Threshold Configuration module

Threshold Configuration module shown in Figure 3-15 sets all the deployment information along with threshold values that can exploit the power of Multi-core environment for the best performance. Threshold Configuration module first sets up the default threshold parameters based on the information available at the design time of the resource prior to deploying the resource under COAF. For example the parameter could be "the default number of threads for MySQL server". Using this default information, System Management module deploys the resource. Once the resource is deployed and initialized with default values, then the resource becomes ready for processing the workload. Then, the Behaviour Information module investigates the behaviour of the resource against the desired design goals, collects the metrics and creates the output logs. These logs are used for correlating the resource behaviour against the resource performance using various parameters. If the performance is optimum then, the threshold parameters are maintained. If the performance is not optimum, then the threshold parameters can be further tuned for the resource. Key components of this module are Configuration server, Correlation map, Correlation engine and Notifier.

3.4.5.1 Configuration server:

Configuration server houses all the deployment configuration parameters along default and threshold values for each type software and hardware resources that are deployed in the system. The default configuration values are provided either by platform provider or provided by the platform administrator based on his/her knowledge of that platform. Configuration server also contains the meta-data about the software binary images for the applications such as MySQL, Apache, etc. Deployment configuration parameters and meta-data information about the binary images together provide the complete information for System Management module to deploy the resource.

3.4.5.2 Correlation map:

Correlation map provides the association between the deployment configuration parameters and the observed parameters at the operating system level. This map is first created using the design time information that is available with the application developer and the system administrator. Some of the mappings in the Correlation map are: (1) number of page faults is

mapped to record size that is returned from the application query; (2) number of Cache miss count is mapped to number of client threads configured in the thread pool. After the correlation map is setup, the associations in the correlation map become the basis for setting up the default values and threshold values for deployment configuration parameters. After the resources are deployed using these configuration values, the system starts to process the workloads. The new knowledge gained by observing the operating system parameters, can further be used to enhance the correlation map or add new parameters to the correlation map.

3.4.5.3 Correlation engine

Correlation engine provides the algorithms and mathematical models to group the events, correlate and finally identify the threshold values. Correlation engine uses the correlation map to mark the parameters that need to be observed at the operating system level. Thus, correlation engine plays a vital role in bridging the design time knowledge with the actual run-time behaviour of the system. Correlation engine takes the input from Behaviour Information module in the form of standard input log files. This log file is parsed and the events relevant to parameters that are mapped in correlation map are extracted. These events are Cache misses, page faults, unaligned memory access, stack overflow, invalid address, etc. These events are then correlated with deployment parameters as identified in correlation map. The deployment parameter can be tuned for the optimum utilization of the Multi-core system. The newly found threshold values of the deployment configuration parameters are packaged and sent as a notification message. If the deployer reset the resource configuration parameters, then correlation engine repeat the cycle, observing, collecting, extracting and correlating, till the system attains the stable state for optimum utilization for desired performance then the threshold values are reset and published in the Notification Framework module. Correlation engine is a web service and hence the implementation can plug in various correlation/clustering algorithms. Thus Threshold Configuration module takes into account design time context of the application developer and the run time context of the actual application, to arrive at the threshold values for the optimum utilization of the Multi-core processor servers.

mapped to record size that is returned from the application query; (2) number of Cache miss count is mapped to number of client threads configured in the thread pool. After the correlation map is setup, the associations in the correlation map become the basis for setting up the default values and threshold values for deployment configuration parameters. After the resources are deployed using these configuration values, the system starts to process the workloads. The new knowledge gained by observing the operating system parameters, can further be used to enhance the correlation map or add new parameters to the correlation map.

3.4.5.3 Correlation engine

Correlation engine provides the algorithms and mathematical models to group the events, correlate and finally identify the threshold values. Correlation engine uses the correlation map to mark the parameters that need to be observed at the operating system level. Thus, correlation engine plays a vital role in bridging the design time knowledge with the actual run-time behaviour of the system. Correlation engine takes the input from Behaviour Information module in the form of standard input log files. This log file is parsed and the events relevant to parameters that are mapped in correlation map are extracted. These events are Cache misses, page faults, unaligned memory access, stack overflow, invalid address, etc. These events are then correlated with deployment parameters as identified in correlation map. The deployment parameter can be tuned for the optimum utilization of the Multi-core system. The newly found threshold values of the deployment configuration parameters are packaged and sent as a notification message. If the deployer reset the resource configuration parameters, then correlation engine repeat the cycle, observing, collecting, extracting and correlating, till the system attains the stable state for optimum utilization for desired performance then the threshold values are reset and published in the Notification Framework module. Correlation engine is a web service and hence the implementation can plug in various correlation/clustering algorithms. Thus Threshold Configuration module takes into account design time context of the application developer and the run time context of the actual application, to arrive at the threshold values for the optimum utilization of the Multi-core processor servers.

3.4.5.4 Notifier

Notifier subscribes to the messages from Behaviour Information module at the start of collection of logs and from Threshold Configuration module at the start and the end of correlation analysis. Notifier generates messages through Notification Framework module about changes in threshold values based on correlated parameters. These notifications enable the System Management module to change the deployment parameters, and thus optimize the Multi-core utilization. Notifier also subscribes to the messages from System Management module after the completion of deployment, undeployment and resetting of configuration parameters.

3.4.6 Core Scheduler module

Core Scheduler module provides the Connection-Orientation for the system. This module works in tandem with Threshold Configuration module and System Management module to achieve the goals of Connection-Orientation. Core Scheduler module has five components namely, Interceptor, Scheduler, ServiceMap, SchedulerState, and Notifier. The Figure 3-16 depicts the interactions happen within Core Scheduler module and with other modules.

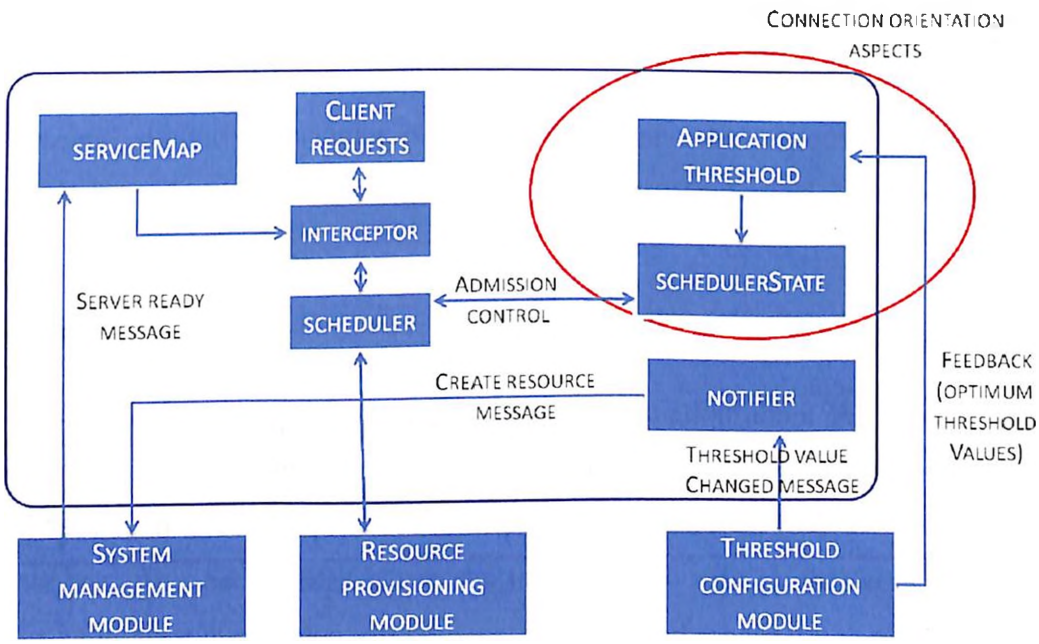


Figure 3-16 Core Scheduler module interactions

3.4.6.1 Interceptor:

Interceptor acts as a proxy for the COAF system. Interceptor intercepts the incoming web-service requests and outgoing responses. Interceptor dismantles the header information, and identifies the destination service along with the associated operation and the destination web service. Similarly, it intercepts the outgoing response and overrides any information at the message header relevant to requested client. Interceptor looks at the ServiceMap to identify service location so that it can dispatch the request to that service. If the destination service is not present in the ServiceMap, then Interceptor rejects the request. If the destination address is valid then the interceptor copies the original server information of the header in SchedulerState and then overrides the portType of the header to the correct server address. After overriding then it hands over the request to the Scheduler. On return from scheduler as a response, Interceptor overrides the source address of the response, so that the response can be sent back to requestor.

3.4.6.2 Scheduler:

Scheduler is the final handler for the incoming request before it is dispatched to actual server that is serving that request. Scheduler maintains a private session object called SchedulerState for each service to establish the Connection-Orientation. The count is incremented when the request arrives, and similarly the count is decremented when the response is sent from the system. The total number of service requests that are currently served is maintained in the SchedulerState.

3.4.6.3 ServiceMap:

ServiceMap is the repository with comprehensive information about the services that are deployed under COAF. The information contained in the ServiceMap is: target Service Name, operations exposed on the service, protocol to dispatch to the service, and server destination to dispatch the service. This ServiceMap is used by Interceptor to validate the service request and if the request is valid, then Interceptor will process the request otherwise it will reject the request.

3.4.6.4 SchedulerState:

SchedulerState provides the vital information for establishing "Connection-Orientation" in COAF. SchedulerState is maintained locally for each web service. SchedulerState is responsible for propagating the application context to the operating system, thereby allowing operating system to perform optimal scheduling. The information maintained in SchedulerState is both static and dynamic information relevant to a web service, and this includes scheduling policy (round robin etc. – if there is a cluster of servers), admission priority policy (Priority A requests will go to server A, all others requests will be dispatched to other server), alternate destination server, number of requests that are currently served, number of requests completed, and the health of the destination server (active, getting repaired, faulty).

3.4.6.5 Notifier

Notifier generates notifications and publishes them through Notification Framework module. The notification is "deploy a new server" when the number of requests reaching the admission policy levels in the current server for a given web service. This message is published to System Management module for deploying the new server. Notifier subscribes to the notifications from System Management module about the completion of deployment of the server. Notifier also subscribes to notifications from System Management module about the non-availability of the server. In those scenarios, the scheduler will update the ServiceMap and SchedulerState classes about the non-availability of the undeployed server.

3.5 ARCHITECTURE BOOTSTRAP AND MODULE INTERACTIONS

Figure 3-17 highlights the components that need to be setup as part of initialising the COAF architecture. Transport senders and transport receivers manage the protocol aspects for receiving web service requests from clients and sending web service responses to clients. SOAP processing model takes care of separating the headers of the web service requests from the body. The information model consists of all the standards that govern the definition of various resources that are deployed on the system; this includes the web services resource frameworks, web services distributed management, web service notification etc, and the global configuration files that are associated with the deployment of the resources. Notification Framework module sets up the event notification management framework. Deployment is about installing all the

resources (hardware and software), initializing and making those resources ready for processing the client request. Once deployed the COAF system is ready to receive the web service requests for processing.

Figure 3-18 depicts the scheduling of web service request and the interactions that happen between the modules to fulfil the processing of this web service request. Client sends the SOAP based web service request to COAF system. Core Scheduler module intercepts all the incoming client requests. The SOAP request contains header and payload, with references to service end points location. This Core Scheduler identifies the service from the header and looks for the service in the ServiceMap. If the ServiceMap does not contain the service, the request is sent back to the client with "service not found" response. If the ServiceMap contains the service, then the scheduler marks the details of the client request that are found in the header to SchedulerState object. Then it identifies the server from ServiceMap where the service to be dispatched to. Core scheduler modifies the header information, and dispatch the request to the server. Once the server completes the processing of the request, the response is created. Core scheduler intercepts the response and overrides the address with original client address that is available in SchedulerState. Every server is capable of processing x number of requests at any point in time. However, if the incoming traffic increases beyond this "x" value, then Core Scheduler requests for a new server. System Management module picks up the request for the new server, and identifies the deployment information including configuration parameter associated with that server. System Management module deploys the server and sends the notification that server is ready for processing. Scheduler adds the new server into ServiceMap and start to schedule the incoming requests to the new server. The number of requests that can be processed in a given server is determined by observing the behaviour of the server at the operating system level. The Behaviour Information module collects the run time behaviour of the server using non-invasive instrumentation techniques. The collected information is refined and relevant events at the operating system that are mapped with deployment parameters are extracted. Correlation engine looks to match the events such as Cache miss/page faults, unaligned memory access, stack overflow, invalid address, etc, with the system parameters and suggest the parameters that could be adjusted. The threshold values are arrived at by modifying the deployment parameters to arrive at the optimum performance at the operating system level.

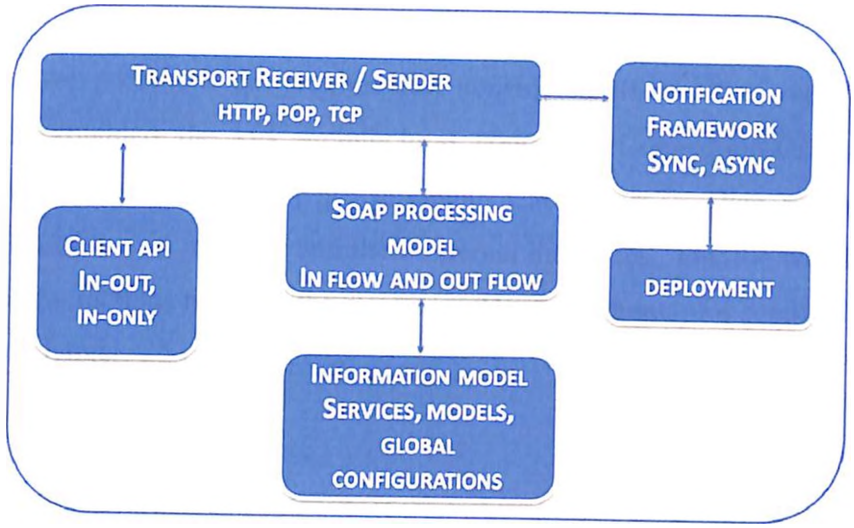


Figure 3-17 Architecture Bootstrapping

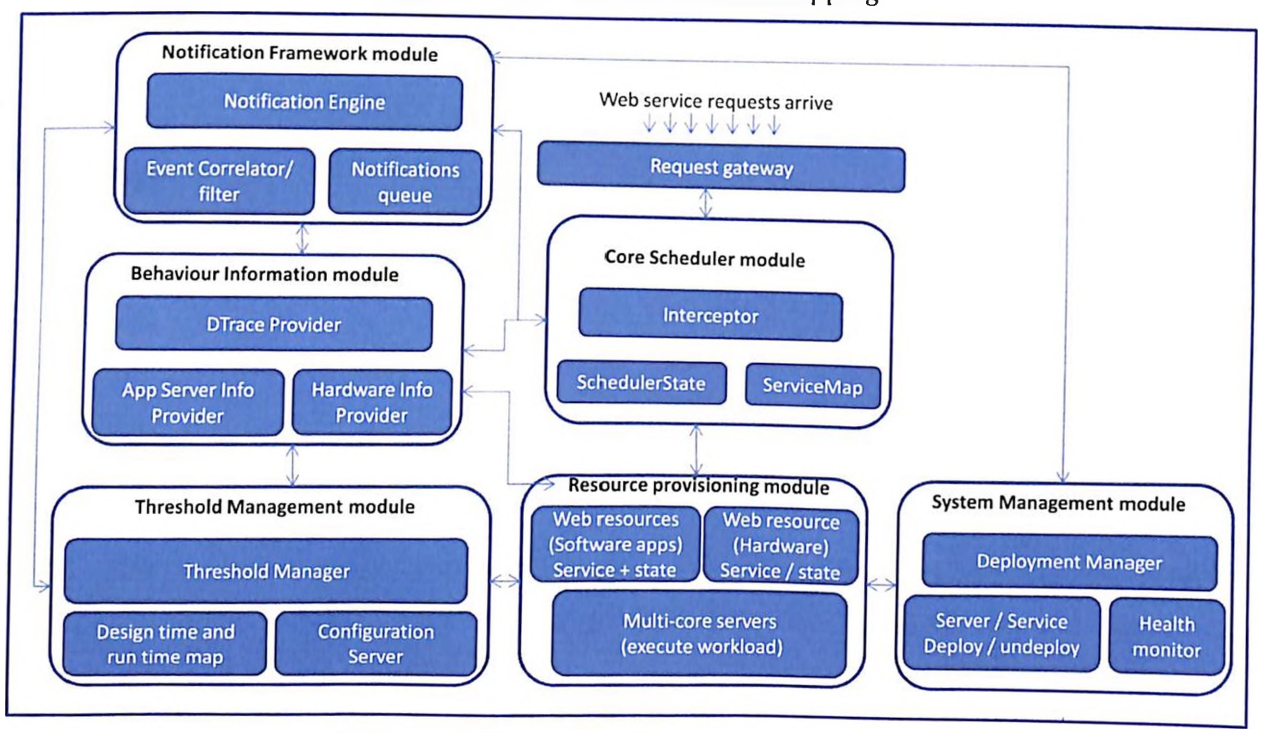


Figure 3-18 Service Scheduling and Module interactions in COAF

These threshold values are then stored in the configuration server. These values get incorporated into deployment configurations by deployer of System Management module. The Notification Framework module provides "publish and subscribe" capabilities for all the modules and assures the receipt of notifications to all recipients. Thus the COAF architecture uses deployment configurations to adjust the behaviour of the Multi-core system without modifying the operating system or the application.

3.6 SUMMARY

In this chapter, we presented the architectural considerations and the overall design of COAF. Next, we explained the key architectural concepts "Connection-Orientation" and "Infrastructure Awareness" along with key measurement "Cache-miss". Then, we discussed the design details for each of the six modules and their external interfaces. Finally, we discussed the initialization and the interactions between the modules. The implementation details of COAF are discussed in the next chapter.

4 IMPLEMENTATION MODEL OF COAF

This chapter details the implementation details of COAF modules, which depict the concepts that are discussed in the previous chapters namely,

1. ability to model and implement the software and hardware as WS-Resource
2. ability to contextually group and dispatch the service requests to appropriate server
3. ability to non-intrusively capture the production logs of the working system
4. ability to leverage the compute power of the core optimally
5. Scalable modular architecture to accommodate various implementations
6. Pluggable architecture to accommodate continuous improvements available in various engines such as correlation engine, threshold engine, etc.

The overall implementation details for six modules of COAF are shown in Figure 4-1 along with the choice of the implementation technology and the details of the interactions between modules is highlighted. The next six sections detail these six modules.

4.1 RESOURCE PROVISIONING MODULE

Resource provisioning module ensures that a compute resource is available for processing an incoming request. The key components of Resource Provisioning module is the stateful resource and the Resource Manager.

4.1.1 Concept of "Resource model"

Resource is an abstraction over hardware like Multi-core server and software like Apache web server that is deployed in the system. This abstraction model helps to plug in a component with a standard interface into the architecture, thus IT enables the interchangeability of implementations and configurations without the need for changing the client code. Resource can have multiple views; this means a resource can have different set of operations for different kinds of clients. Resource is a stateful resource; this means a state of the resource can be set and queried at any point in time; this ability is useful for managing the resource. Resource is standards based interface; the clients interact with the resource using standard interfaces, without knowing the internal implementations of the resource.

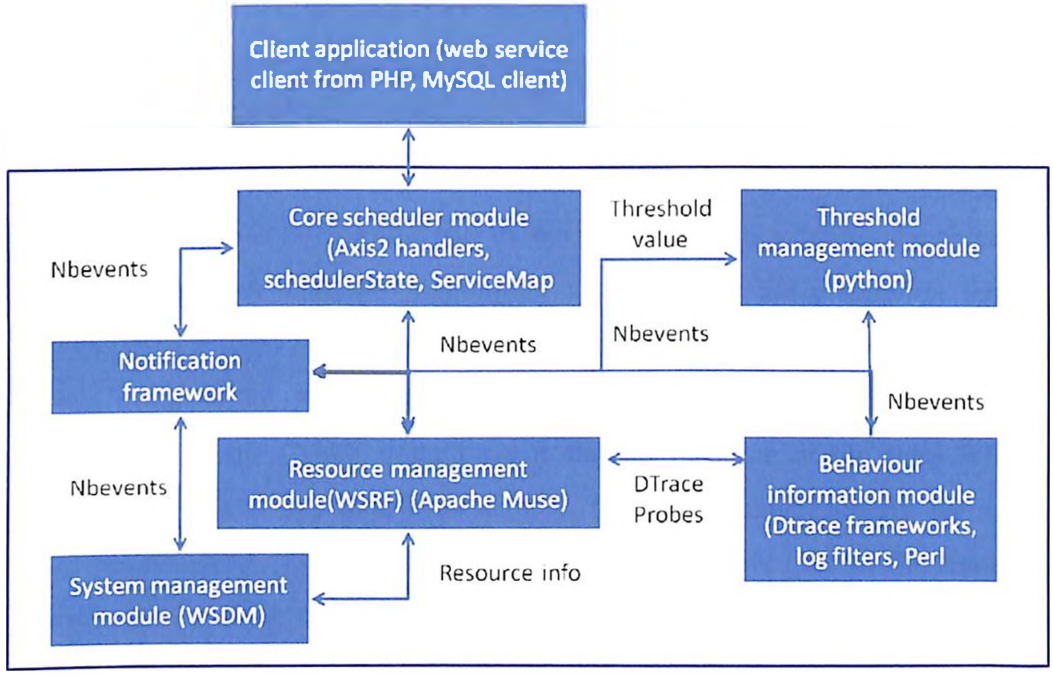


Figure 4-1 Overall Architecture Diagram for COAF

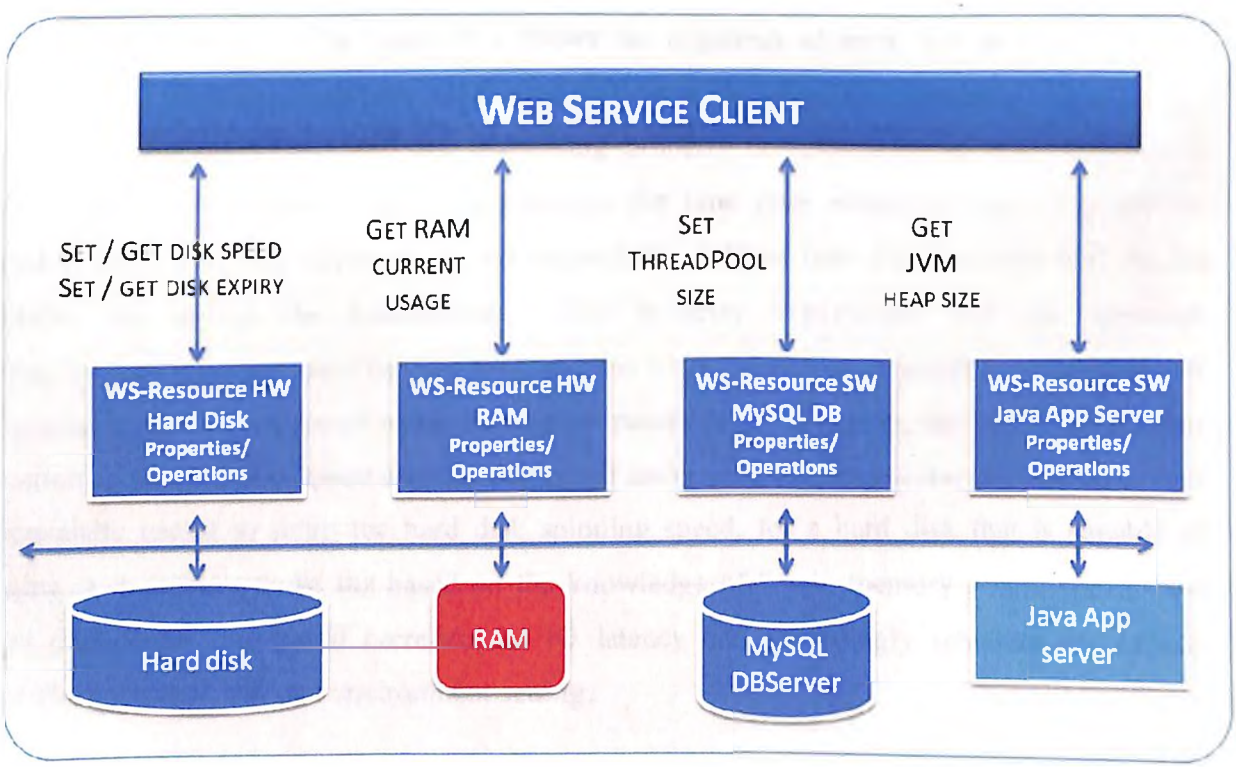


Figure 4-2 Resource Modelling

4.1.2 Implementation details for COAF resource

Web services Resources Framework (WSRF) [179] is used for modelling a COAF resource. Apache Muse [15] is used as the implementation container for WSRF standard. WSRF standard supports an implementation of a stateful resource of the type called WS-Resource (WSR). WSR is a web service standard defined as the part of WSRF. It natively supports the object orientation model thus implementing state and operations. Additionally, WSR supports the concept of views. These two concepts namely the object orientation and the view, enables the process of externalizing the resource provisioning aspects from its internal implementation. All the resources deployed in the COAF system are a stateful resource of the type WSR and each resource implements three key interfaces namely, (1) WSResourceProperty for managing the properties of the resource, (2) WSResourceLifetime to mange life time of the resource, and (3) WS-Notification to manage the Notification process between web services. All these three interfaces are mandatory so that the basic needs of manageability are addressed. The diagram Figure 4-2 depicts the resource model in the COAF framework. As shown in the Figure 4-2, both hardware (hard disk and RAM) and software (MySQL DB server and Java App server) are modelled as resources. The Figure 4-3 shows the depiction of hard disk as a web service representation in XML language. For hard-disk as a resource, the operation name on the resource is `SetExpiryTime`, and the underlying property corresponding to this operation is `ExpiryTime`. This property `ExpiryTime` defines the time upto which the hard disk will be available for storing the information, and beyond this defined time the hard disk will not be available for storing the information. This property `ExpiryTime` and the operation `SetExpiryTime` together could be used for scenarios when the hard-disk needs to be taken out of the production; for the sake of either backup or maintenance. Similarly, for hard disk, another operation is `SetHardDiskSpeed` and the associated underlying property is `HardDiskSpeed`. This is especially useful to setup the hard disk spinning speed, for a hard disk that is capable of running at multiple speeds; the based on the knowledge of "main memory access speed" and "hard disk speed" we could correlate the IO latency and accordingly schedule the request dispatch that would suit that environment setting.

```
<wsrf-rp:GetResourcePropertyDocumentResponse>
<Disk xmlns="http://COAF/harddisk">
  <operation_properties>
    <ExpiryTime>2010Jan10-2200-EST</ExpiryTime>
    <HardDiskSpeed>4000</HardDiskSpeed>
  </operation_properties>
```

Figure 4-3 Code Sample – Resource Properties

```
<MYSQL_DB_Server xmlns="http://COAF/MySQLdbserver">
  <operation_properties>
    <MaxThreadPool>40</MaxThreadPool>
    <MinThreadPool>20</MinThreadPool>
  </operation_properties>
<wsdl:portType name="MySQLDbServerService"
  wsrf-rp:ResourceProperties="MySQL_DB_Server ">
    <wsdl:operation name="GetMaxThreadPool">
      <wsdl:input message="wsrf-rpw:GetMaxThreadPoolRequest" . . />
      <wsdl:output message="wsrf-rpw: GetMaxThreadPoolResponse" . ./>
    <wsdl:operation name="GetMinThreadPool">
      <wsdl:input message="wsrf-rpw:GetMinThreadPoolRequest" . . />
      <wsdl:output message="wsrf-rpw: GetMinThreadPoolResponse" . ./>
    . . .
    </wsdl:operation>
  . . .
  </wsdl:portType>
```

Figure 4-4 Code Sample – Resource Map for an application MySQL server Properties

Similarly, the software can be modelled as WS-Resource, and the Resource properties and operations can be used to setup the software deployment configuration. For example, the XML definition for MySQL database server as WS-Resource is shown in code sample Figure 4-4. There are more than two hundred deployment parameters to configure the MySQL server; out of these parameters, two properties namely `MaxThreadPool` and `MinThreadPool` are shown in the figure that are used for setting up maximum and minimum thread pool size for MySQL server respectively. To view the current settings of these two parameters, the operations `GetMaxThreadPoolRequest` and `GetMinThreadPoolRequest` are used. Similarly all the configuration parameters for a given resource can be set queried using the above methods. This way of externalizing a property and defining it using an XML definition helps in querying and modifying a resource at real time; this externalization of operations and property helps to store the state externally, thereby preserving the temporal behaviour of the system; these stored values are then used for correlating the configurations parameter set at that point in time with the behaviour of operating system. This ability to query the deployment configuration helps us to understand the infrastructure setting in a point in time, which is one of the key "Infrastructure Awareness" aspect; similarly the ability to setup the deployment configuration on a resource helps us to modify the configuration parameter value for optimum threshold values.

4.1.3 Resource Manager

Resource Manager is a web service. It provides the interface to external world for accessing the resource. Once the System Management module deploys the resource, Resource Manager starts to dispatch the requests to the resource. Clients come through Core Scheduler module into Resource Manager for accessing the resource. Resource Manager acts like a resource factory; thus, Core Scheduler is able to call the Resource Manager to establish the Connection-Orientation with the resource for dispatch the request. This ability of the Resource Manager for establishing the Connection-Orientation is an important requirement for the COAF based enterprise application.

4.2 BEHAVIOUR INFORMATION MODULE

Behaviour Information module provides the run-time behaviour information of the system. This is done by instrumenting the system with appropriate probes on the resources and the collected information is provided to the subscribers in the form of notifications. This module

consists of Probe instrumentation framework, Output provider and Notifier. Probe instrumentation framework provides the mechanism to observe and measure various parameters and these measurements are log files. Output provider converts the generated logs into meaningful output files, which the Threshold Configuration module can consume and do further analysis. Notifier informs the availability of log files to external systems.

4.2.1 Probe Instrumentation framework:

Probe instrumentation framework shown in Figure 4-5 consists of non-intrusive information providers that can extract both static and dynamic real-time information of the system behaviour. This information could be fine grained or coarse grained; fine-grained at the level operating system and answer the questions such as which process was running on CPU, the number of Cache misses, how many times the process was interrupted and rescheduled etc.; coarse grained at the level of web service, and answer questions such as how long it took to execute a web service operation. DTrace [103] framework is used as the non-intrusive in-production instrumentation framework for COAF. There are other instrumentation systems such as SystemTap [216], and LTTng [148], are available for observing the system at the operating system level as shown in Appendix I. DTrace has direct support for enterprise class platforms such as Apache, MySQL, and PostgreSQL. There are two steps involved in setting up the probe instrumentation framework as shown in Figure 4-6; the first is to setup and configure the metrics for the probe infrastructure, and the second is to capture the logs that are emitted from the probes and store them in the appropriate directory, so that the output provider can fetch these logs to create meaningful output files.

4.2.2 Output provider

Output provider as shown in Figure 4-7 is a set of adaptors to convert the input files obtained from Probe instrumentation framework into meaningful output files. DTrace framework creates a set of log files from the probes. Probes dump the raw data into these log files. These raw data files contain all the information including the information related to processes that are executed for DTrace. This raw data is not directly useful for extracting the information that is needed. These log files need to be refined for extracting meaningful information for correlation. From these log files the "parameters of importance" (for example number of Cache misses) are extracted using parameter specific adaptor.

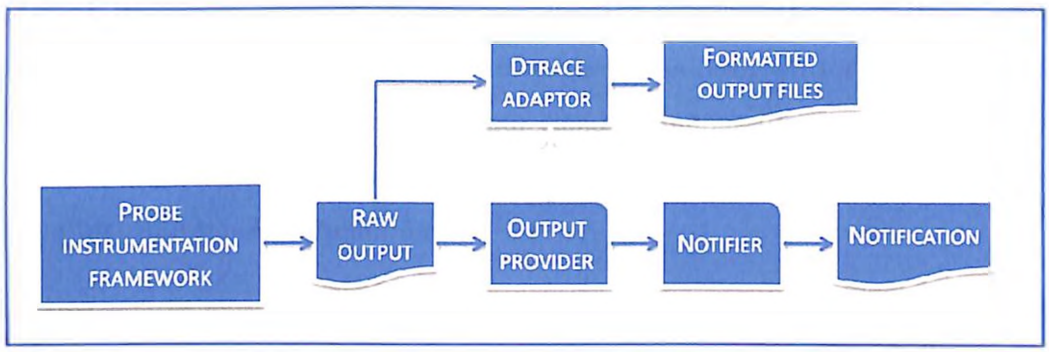


Figure 4-5 Behaviour Information module - Probe Instrumentation

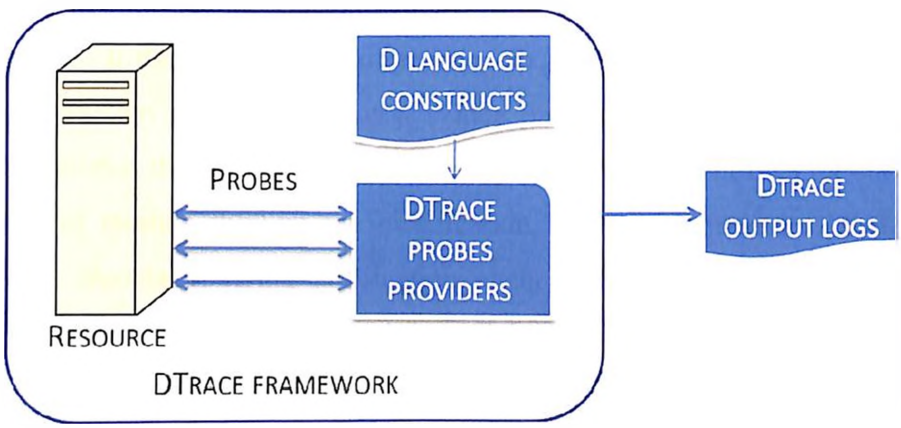


Figure 4-6 DTrace Framework – Probe output logs

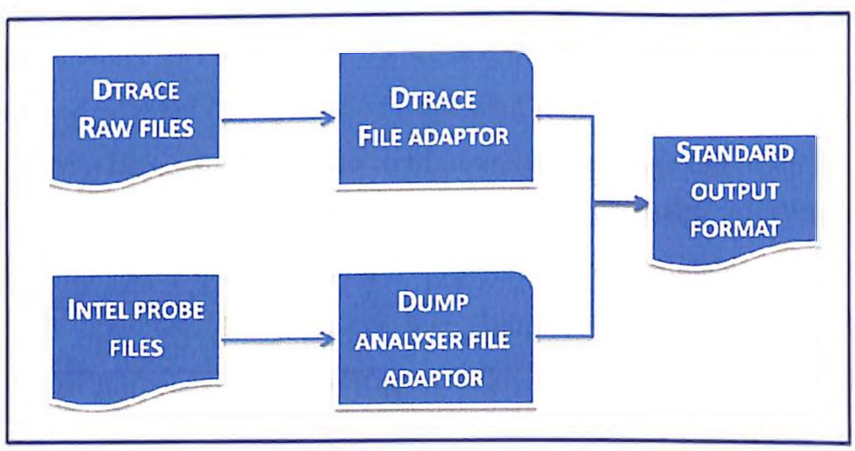


Figure 4-7 DTrace Output Provider

For each parameter, there is an adaptor. Set of these adaptors together called "output provider". There is a standard format and template that is designed for each parameter. Using the adaptor the input files are converted into standard output files containing the values for "parameters of importance". Output provider has multiple transformation adaptors to convert the DTrace log files into standard output format.

4.2.3 Notifier

Notifier performs the role of subscribing to messages from other modules and publishing the notification of Behaviour Information module to other modules. Notifier subscribes to request for starting the probe information collection start with necessary additional parameters for information collection. It then passes this information to probe instrumentation framework for the start of the information collection process. Once the output files are created by output provider, Notifier publishes the notification about the availability of output files for correlation to threshold management module through the Notification Framework module. Notifier is an implementation of a NaradaBrokering notification agent and publishes the WS-Notification message to the Notification Framework module. This published message is picked up by Threshold Configuration module for analysis and correlations.

4.3 SYSTEM MANAGEMENT MODULE

System Management module implements the functions required to deploy and undeploy a resource in COAF framework. Web Services Distributed Management (WSDM) [178] standard based model is used for designing the System Management module. WSDM is based on web services standard, and provides a standard way to discover and manage the resources. It allows the wrapping of resource definition models, such as CIM, SID, SNMP etc. WSDM coexists along with WSRF and WSN specifications and thus provides an easy implementation mechanism for System Management module. There are two key components of the System Management module namely Deployment manager and Notifier. Deployment manager deploy, undeploy and manage the resources. Notifier helps in co-ordinating the activities of System Management module with other modules of the system.

4.3.1.1 Deployment manager

Deployment manager performs the role of deploying and undeploying the resources. Deployment manager receives the notification through notifier from Core Scheduler module for a new resource requirement for scheduling. Deployment manager then unwraps the notification to identify the component that needs to be deployed. Deployment manager extracts the configuration information associated with the resource from the configuration server. There is a dedicated deployment mechanism for every component. For example, the class DeploymentEngine (belongs to package org.apache.axis2.deployment) is used deploying Axis2 software resource. Similarly MySQLd software process can be deployed using the descriptor as below:

```
<?xml version="1.0" encoding="UTF-8"?>
 <server>
  <mbean code="com.MySQL.management.jmx.jboss.JBossMySQLdDynamicMBean"
     name="MySQL:type=service,name=MySQLd">
  <attribute name="datadir">/tmp/xxx_data_xxx</attribute>
  <attribute name="autostart">true</attribute>
  </mbean>
 </server>
```

Similarly, the deployment manager undeploys the resources using the undeployment methods relevant to that resource and sends the notification to Core Scheduler module about the non-availability of the resource for further processing.

Deployment Manager maintains the lifetime information of the resource. These are: ResourceID, ResourceName, ResourceCount, ResourceDeployedOn, ResourceCreatedTime, ResourceLifeTime, ResourceSpecificPropertiesRefID, ResourceViewsSupportedCount, ResourceEndPointReference and ResourceDeploymentConfigurationVersion. XML file format is used as the storage format. ResourceViewsSupportedCount helps in monitoring the resource, if there are multiple views defined for the resource, and ResourceSpecificPropertiesRefID helps in investigating additional properties of the resource. Deployment manager additionally implements the seven attributes to monitor the health of the resource. They are ResourceID, ResourceState, ResourceDescription, ResourceOperationalStatus, ManageabilityCharacteristics, CorrelatableProperties and Metrics. These properties are base attributes provided by WSDM framework. ResourceID is the Identity of the resource. It is globally unique; neither mutable nor

modifiable; and can be correlated: if two reported ResourceIds are identical, then they refer to the same manageable resource. ResourceState is the state of the resource, exposed through the State capability. This is implemented by each resource. ResourceDescription is about the resource, its version etc. This information is used to identify the resource with its deployment configurations, if it needs to be redeployed or recovered. ResourceOperationalStatus is about the high-level health of the resource such as Available, PartiallyAvailable, Unavailable or Unknown. If Unknown, the notification is sent to Notification Framework module so that scheduler will stop scheduling the service requests to that resource. ManageabilityCharacteristics, which exposes a list of ManageabilityCapability elements that are supported by the resource; CorrelatableProperties capability exposes a list of properties whose values are useful when determining whether two different ResourceIds from two different manageability providers actually refer to the same manageable resource. Metrics S defines the performance and operation of resource. WSDM defines some metrics for a Web service resource and allows all resources to define any suitable and relevant metrics.

4.4 NOTIFICATION FRAMEWORK MODULE:

Notification bridges all the modules of COAF as seen in Figure 4-8. The Notification Framework module needs be a scalable messaging system that can interact with all the modules. Following are the requirements envisaged for the Notification Framework module.

1. Ability to support publish / subscribe messaging model, so that COAF sub-systems can carry out their tasks and need not react to external events as long as they are not notified.
2. Ability to work in isolation mode and in clustered mode.
3. Assured message delivery within the defined service level agreements.
4. Ability to recover from failures based on its own state.
5. Ability to support for multiple-communication protocols.

There are three design models available to implement Notification Framework module, namely WS Notification, WS Eventing and Enterprise Service Bus. The following implementations were evaluated for the Notification Framework module

a) Apache Muse [15], an implementation of the WS Notification family of specifications, also supports Web Services Resource Framework (WSRF and Web Services Distributed Management (WSDM)

b) Apache Pubscribe [16], an implementation of the WS Notification family of specifications

c) ServiceMix [17], Enterprise Service Bus with support for WS Notification

d) JBossWS [191], an implementation from JBoss that supports WS-Eventing

e) NaradaBrokering [181], custom implementation for WS-Eventing

f) WS-Messenger [115], custom implementation that supports both WS-Notification and WS-Eventing specifications.

NaradaBrokering version 3.3.0, from Indiana University is used for implementing the messaging framework. NaradaBrokering supports WS Evening in distributed environment. NaradaBrokering is written in Java and has been tested on Windows, Linux and Solaris based systems. NaradaBrokering provides messaging middleware, high availability, clustered representation of messages, support for various protocols, and implementation of Web service standards.

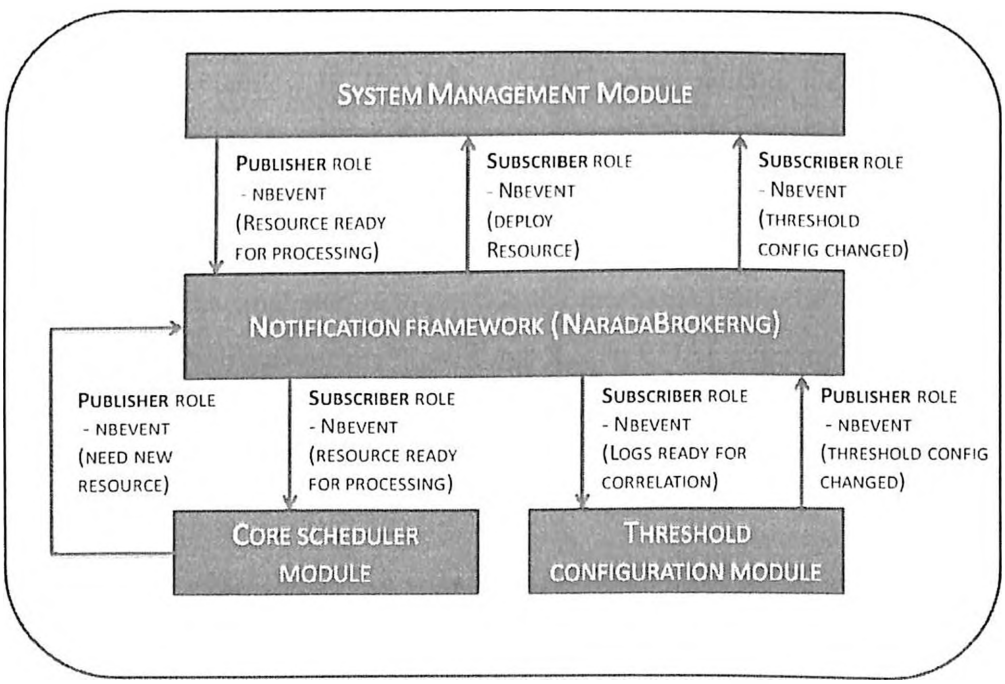


Figure 4-8 Flow Diagram of Notification Framework module

In the NaradaBrokering framework, a COAF module can be a publisher or a Subscriber or both. All the modules of COAF participate and use the Notification Framework module by sending nbevents among themselves. Broker service is first initialized using the configuration stored in the file called ServiceConfiguration.txt and the clientService class. The code snippet shown in Figure 4-9 sets up the client services. Here entityID represents any participant in the NaradaBrokering system. Entities in COAF system represent Core Scheduler module, Threshold Configuration module, System Management module, Behaviour Information module, and Resource Provisioning module. Using the clientService every participant in NaradaBrokering gets their own identity. Once they get the identity, they can play the roles of either Subscriber or Publisher. The next step is to setup the communication channel between the client service and the broker, using any of the protocol supported in NaradaBrokering. This is shown in code snippet Figure 4-10 that represents the setting up and initialization of communications protocol with the broker using TCP. The next step is to setup both the subscriber role (consumer role in NaradaBrokering) and the publisher (producer role in NaradaBrokering) as shown in Figure 4-11. The next step is to setup the event processing logic in nbEvent as seen in Figure 4-12. NbEvent is the implementation of NaradaBrokering event service. It comprises of headers, content descriptors and the payload encapsulating the content. The nbEvent's header provides the information pertaining to the type of the event, unique identification for the event, timestamps, dissemination traces and other quality of service related information pertaining to the event. The content descriptors for the nbEvent describe information pertaining to the encapsulated content. The content descriptors and their values collectively comprise the event's content synopsis. The final step is to publish the messages using the eventproducer as shown in Figure 4-13. The Notification Framework module in COAF can similarly use NaradaBrokering clientService to support other messaging mechanisms such as JMS queues namely topics, and create the roles such as topicProducer and topicConsumer.

```
Int entityID = 1000;
String config = "/COAF/config/Narada/ServiceConfiguration.txt";
SessionService.setServiceConfigurationLocation (config);
ClientService clientService = SessionService.getClientService(entityID);
```

Figure 4-9 Setting up the client services

```
Properties props = new Properties();
props.put("hostname", "localhost");
props.put("portnum", "8050");
clientService.initializeBrokerCommunications(props, "tcp");
```

Figure 4-10 Communications protocol initialisation using TCP

```
// setting up event consumer role
int profileType = TemplateProfileAndSynopsisTypes.STRING;
Profile profile = clientService.createProfile(profileType,
"CoreSchedulerModule");
consumer.subscribeTo(profile);
// setting up Producer role
EventProducer producer = clientService.createEventProducer();
producer.setSuppressRedistributionToSource(true);
producer.generateEventIdentifier(true);
producer.setTemplateId(12345);
producer.setDisableTimestamp(false);
```

Figure 4-11 Setting up subscriber role and producer role in NaradaBrokering

```
public void onEvent(NBEvent nbEvent) {
String synopsis = (String) nbEvent.getContentSynopsis();
COAF.write (moduleName + "Received NBEvent {" + synopsis + "} "
+ new String(nbEvent.getContentPayload()));
}
```

Figure 4-12 Event Processing Logic in NaradaBrokering

```
int eventType = TemplateProfileAndSynopsisTypes.STRING;
String synopsis = "event occurred";
byte[] payload;
NBEvent nbEvent = producer.generateEvent(eventType, synopsis, payload);
producer.publishEvent(nbEvent);
```

Figure 4-13 Publish the messages using the EventProducer

4.5 THRESHOLD CONFIGURATION MODULE

Threshold Configuration module performs the role of establishing the feedback to the COAF system for achieving the optimum performance on the Multi-core server through threshold values. It computes and manages the threshold values for various deployment configuration parameters for every component that is deployed in the COAF based system. Threshold Configuration module has three components namely, Configuration server, Correlation map and Correlation engine.

4.5.1 Configuration server:

Configuration server stores and manages the various deployment configuration parameters for every component in the system. It stores the configuration information in the file. There is a specific file created for each component. These files are stored in a pre-defined directory path so that these files can be fetched for configuration management. Each deployment configuration is stored in the following nomenclature

- `<Component name>-<version number>-<core configuration>Setup.txt.`

For example, for MySQL version 5.0.33, for 4 cores configuration, the file name is

- `MySQL-5.0.33-4coreSetup.txt`

Similarly for Apache web server, version 4.0.3, 17 cores configuration, the file name is

- `ServiceMix-4.0.3-17coreSetup.txt.`

Within these files, the deployment values are stored as name value pair. For example, MySQL there are 290 parameters that are available for changing the behaviour of MySQL. Out of these 290 parameters, 165 are changeable at runtime. Among these 290 parameters 225 parameters need the restart of the MySQL server for the configuration to be activated. In the context of thread, cache and memory management by MySQL there are 23 parameters that can affect the behaviour of the application performance as shown in Appendix VIII. This concept of configuration file and the parameters can be extended to all other application platforms like Apache web server, Tomcat server etc.

4.5.2 Correlation map

Correlation map is the map between two parameters for a server in a particular core configuration. It consists of two parts namely, the observable parameter at the resource level ("number of Cache misses) and the deployment parameter impacting that observable parameter

(for MySQL the parameters such as thread_cache_size, bdb_cache_size, shared_memory etc.). This is implemented as both "one to many", and "many to many" relationship. There are two implementation methods provisioned in the architecture. They are name-value pair in the text file, and a table in the relational database system. Both these implementations are accessible with a web service interface and deployable as web services.

4.5.3 Correlation engine

The input data for variance analysis comes from the Behaviour Information module. Correlation engine is a web service that abstracts the correlation process using the input data. The architecture provides the ability to use correlation engine as a web service. In the test bed prototype, a custom correlation engine implementation using Excel ANOVA services [160] is used. This implementation reads the XML file to read the number of groups and number of members for each group.

4.6 CORE SCHEDULER MODULE

The Core Scheduler module shown in Figure 4-14 implements the concepts of Connection-Orientation and contextual dispatch using the three components namely, ServiceMap, SchedulerState and Interceptor. Apache Axis2 framework is used to implement this module. Axis2 provides the readymade implementations such as handlers, phases and flows, which directly support the implementation needs for Core Scheduler modules.

4.6.1 Axis2 handlers, Phases and Flows, and contexts

Axis2 handler is the smallest execution unit that intercepts the incoming service request from a client. The handler can both read and write to an incoming SOAP message. These handlers can be grouped to form the chain. There are default handlers provided by Axis2 for standard operations such as URL routing etc. These handlers are defined in the XML configuration file called Service.xml. There is a standard implementation of Invoke () method is required for every handler, and the return value of the invoke method allows the Axis2 engine to carry the execution to the next handler as defined in the service.xml.

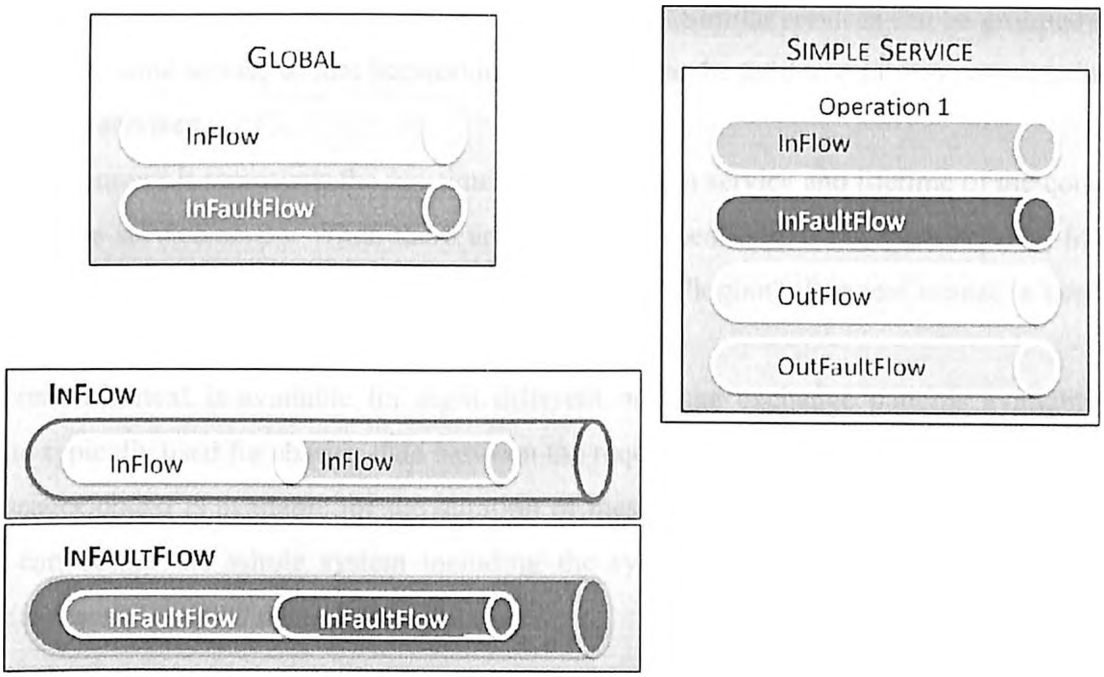


Figure 4-14 Core Scheduler

The concept of phase and flow enables the implementation of dynamic ordering of handlers. Phase is a collection of handlers; flow is a collection of phases. Axis2 engine invokes method of each phase in a given flow, and then the phase will sequentially invoke all the handlers in it. A phase has different phase rules like phaseFirst and phaseLast and conditions like pre and post condition checks. There are three kinds of deployment configurations namely axis2.xml, services.xml and module.xml. And there are four flows, namely inflow, outflow, infaultflow, and outfaultflow helps in organizing the service request flow. Axis2 provides support for five levels of session contexts, makes the installation of resources and services in various contexts. They are: ConfigurationContext, ServiceGroupContext, ServiceContext, OperationContext and MessageContext.

ConfigurationContext – It is a run-time representation of the whole system and exists for the lifetime of the system. ServiceMap can be kept in this ConfigurationContext, so that the lookup can happen in memory.

ServiceGroupContext – It is used to store and share data across services. The lifetime of this context depends on the service scope. If the service scope is "application", then the lifetime will be similar to the system's lifetime. However, if the service scope is "request", then there will be a

ServiceGroupContext created for each and every invocation. Similar services can be grouped and deployed in the same server, so that ServiceGroupContext can be used to establish connectedness among similar services.

ServiceContext: It represents the run-time data for a given service and lifetime of the context depends on the service scope. When there are multiple invocations of the same service, like a service with three operations: "login", "doSomething", and "logout", ServiceContext is used to share the data across operations.

OperationContext is available for eight different message exchange patterns available in Axis2 and typically used for sharing data between the request and the response.

MessageContext is available for the duration of message transport from receiver to sender, and we can access the whole system including the system runtime, global parameters, and property service operations using this context.

4.6.2 ServiceMap

ServiceMap maintains the information about the location of services, where the client request needs to be dispatched. There are two options provided in the architecture to implement ServiceMap. It can be a global implementation across all services in the COAF infrastructure or it could be local implementation to a particular service group. ConfigurationContext is used to hold the ServiceMap for the global implementation and the ServiceGroupContext is used to hold the ServiceMap for the local implementation. The ConfigurationContext holds ServiceMap and SchedulerState data structure in memory, so that the lookup for service happens quickly before dispatching the incoming request to an appropriate core. ServiceMap contains the following name-value pair information, where the serviceName represents the web service and the ServerID-WSA represents the port ID of the server address where the web service is deployed.

serviceName	ServerID-WSA
getAccountSummary	varchar(30)
updateBillingInfo	bigint(20)

4.6.3 SchedulerState

SchedulerState is used for establishing the Connection-Orientation and admission control before dispatching the client request to the server. The SchedulerState maintains the data about the current workload in the server using three values. They are (1) Optimum number of requests, (2) Number of requests currently served and (3) Maximum number of requests. The "optimum

number of requests" and "maximum number of requests" are derived from threshold values calculated from Threshold Configuration module. Number of requests currently served is managed by the Core Scheduler module. Whenever the number of requests currently served approaches the optimum number of requests, the scheduler module notifies the need for new server, which is picked up by System Management module to deploy a new server. SchedulerState is implemented at the ServiceContext or GroupServiceContext level of Axis2 hierarchy, and the choice of the schedule depends on whether a single service is deployed or groups of services deployed as shown in the Figure 4-15. LifeCycle interface ("org.apache.axis2.service.Lifecycle") is implemented with two key methods namely init() and destroy(), where the service_request_currently_served_count in the SchedulerState data structure is updated. These two methods *viz.*, *init* and *destroy* are automatically called by Axis2 run time when a session for the service in context starts and ends respectively. If the service_request_currently_served_count approaches the threshold value, then the Core Scheduler requests the System Management module to prepare next server for sending the requests. If the "service_request_currently_served_count" goes beyond the threshold value then dispatch handler in the Axis2 will not allow the service to pass through the handler chain. The code snippet in Figure 4-15 represents the design of the service_request_currently_served_count.

4.6.4 Interceptor

Interceptor is the gateway for the COAF architecture. It intercepts the incoming requests and based on ServiceMap and SchedulerState Information, routes the request to appropriate server. Similarly, it intercepts the outgoing responses to override the correct destination address. It is implemented using Axis2 handler mechanism as shown in Figure 4-16. Interceptor is an implementation of AbstractHandler class called COAFInterceptor. COAFInterceptor class lives for the life of the context within which the handler is executing. COAFInterceptor has two important methods namely lookup_ServiceMap, and update_SchedulerState, among other methods. Lookup_ServiceMap method is used to verify the validity of the server address mentioned in the request header. If the address is valid, then update_SchedulerState method is used to remember the original address of the request where the response needs to be dispatched to. The code snippet in Figure 4-17 shows the high level implementation of COAFInterceptor.

```
public class COAF_service implements Lifecycle {
    public void init(ServiceContext context) throws AxisFault {
      // increment the service_request_currently_served_count
      serviceContext.getscheudlerstate{}.increaseCount(service_request_cur
rently_served_count);
    }
    public void destroy(ServiceContext context) {
      // decrement the service_request_currently_served_count;
      serviceContext.getscheudlerstate{}.decreaseCount(service_request_cur
rently_served_count);
    }
}
```

Figure 4-15 Scheduler State - Service Request Implementation

```
public class COAFInterceptor extends AbstractHandler
{
    public COAFInterceptor () {}
    public InvocationResponse invoke(MessageContext msgContext)
throws AxisFault {
        //interception logic goes here
      lookup_ServiceMap();
      update_SchedulerState();
        return InvocationResponse.CONTINUE;
    }
}
```

Figure 4-16 Interceptor Implementation

```
public class COAFScheduler extends AbstractHandler
{
    public COAFScheduler(){}
    public InvocationResponse invoke(MessageContext msgContext)
throws AxisFault {   // Scheduling logic goes here
      read_scheduling_policy();
        if less_than_threshold() {  dispatch_request(); }
        else { Send_notification_for_new_server(); }
           return InvocationResponse.CONTINUE;
    }
}
```

Figure 4-17 Scheduler Implementation

4.6.5 Scheduler

Scheduler ensure the Connection-Orientation and admission control for the COAF architecture. It is implemented as shown in Figure 4-17 using Axis2 handler mechanism and configured as the last handler before it is dispatched to the actual server. Scheduler is an implementation of AbstractHandler class called COAFScheduler. This COAFScheduler class lives for the life of the context within which the handler is executing. COAFScheduler has two important methods namely read_scheduling_policy, and dispatch_request among other methods. The read_scheduling_policy is used for admission control. The request is sent to the server based on the "number of current requests that are processed" information available in SchedulerState. If the number of current requests less than the threshold values, then the request is marked for dispatch and dispatch_request method is called to dispatch the request to the appropriate server. If the number of current requests is nearer to the threshold value, then a request for deploying a new resource is sent to Notification Framework module.

4.7 SUMMARY

This chapter identifies the implementation for each of the six modules of COAF. For every module, the choice of the technology for the implementation and the high level design details are presented. The implementation model of service map alongside Axis2 handlers and session contexts provides the ability to contextually despatch service requests thereby achieve Connection-Orientation. The Connection-Orientation affects the performance of the application positively, and improves the compute capacity associated with a Multi-core deployment. This improvement in performance can be verified with a test bed with non intrusive measurement mechanisms discussed in the chapter 5. Similarly, using the non-intrusive instrumentation mechanism of DTrace, we are able to observe the operating system parameters. This ability to measure, monitor and correlate with deployment parameters of the system provides the Infrastructure Awareness to the system.

The pluggable nature of COAF facilitates continuous improvement in the implementations of various components of all the six modules. COAF implementation can be verified using a test bed.

5 IMPLEMENTATION OF A COAF TEST BED

This chapter demonstrates the reference implementation detailed in chapter four with a typical test bed that is based on Niagara based Multi-core server. Each of the six modules and their interfaces manifest in the implementation of COAF, as shown in Table 5-1.

Table 5-1 Implementation map of modules on a test bed.

Module	As mapped on the test bed
Resource Provisioning	• MySQL server, Apache server, etc. are modelled as the software resources. • 12 different core configurations (single core, two cores, four cores, eight cores etc.) are modelled as the hardware resources.
Behaviour Information	• Standard set of DTrace probes and custom developed extraction utilities.
System Management	• Apache deployment framework, XAMPP deployment frameworks are used for deployment and health monitoring of the resources.
Threshold Configuration	• Number of MySQL threads is used as the correlated parameter and the Correlation engine is based on Microsoft Excel services.
Core Scheduler	• Axis2 handler is used as interceptor. ServiceMap and SchedulerState are implemented to dispatch the client request.
Notification Framework	• Notifications are implemented in NaradaBrokering nbevent.

The experiments are conducted in five phases to test the hypotheses identified.

The focus for each phase is depicted in the Figure 5-1. The frameworks, setup methodologies, and tools used for these experiments are consistent throughout. The details of the choice of the probes, and probe environment are highlighted in Appendix II. The user space performance is measured at the application level and, the kernel space performance is measured at the operating system process level for all the five phases.

These five phases are summarised below:

Phase I deals with three questions specific to Multi-core deployment:

1. ability to deploy an application across various core configurations,
2. linear scalability in performance for increasing core configurations, and
3. kernel parameters that need to be identified to reflect the application behaviour.

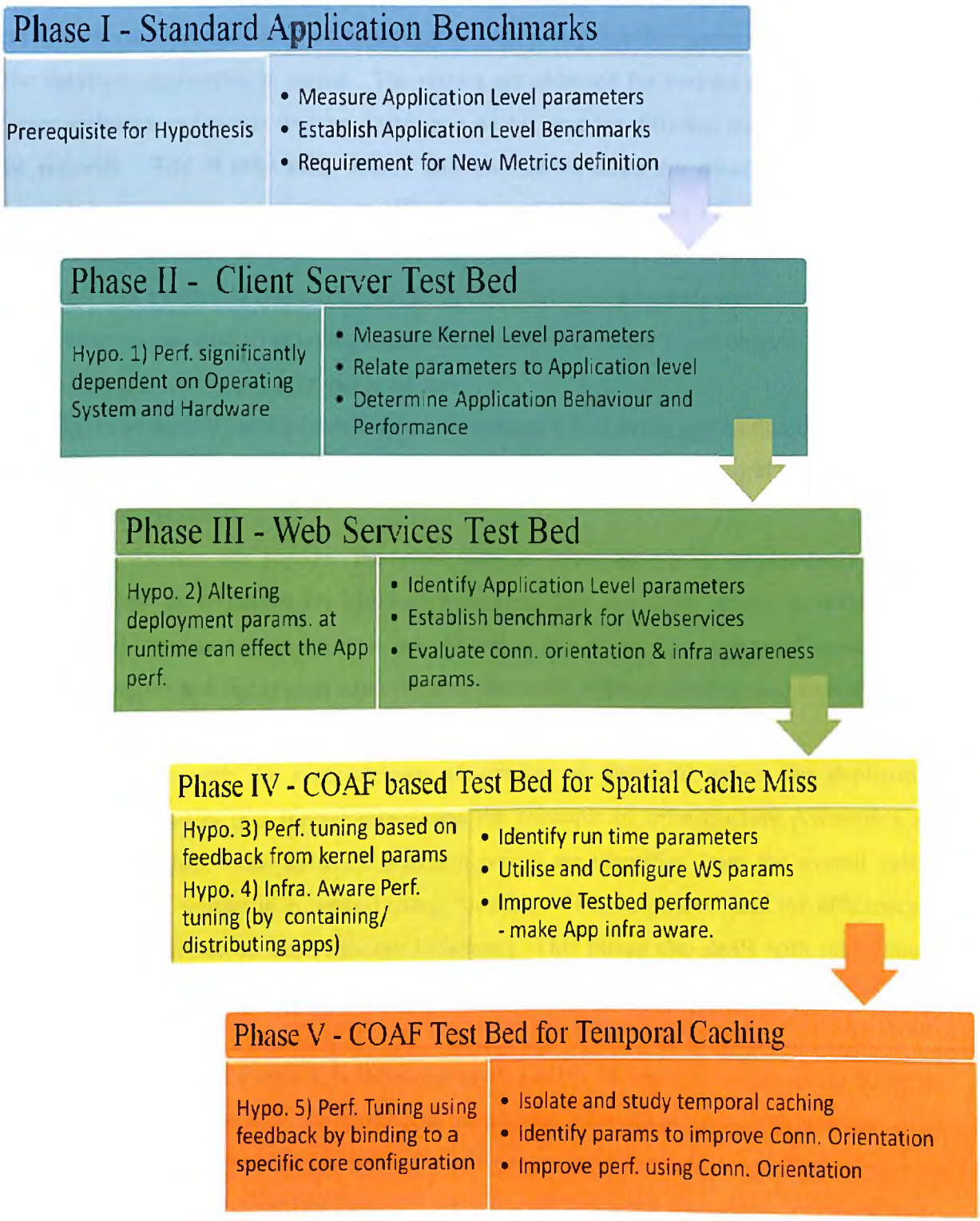


Figure 5-1 Towards realisation of COAF - Five Phase of Experiments and Focus of each phase

Phase I consists of a test bed with a set of ten test cases. These ten test cases represent the operations of a database application from create, read, update and delete operations, to complex select operations. The test cases are organized in such a way that the number of records returned by the database application is varied. The results are obtained for various core configurations, different architecture layouts such as 32 bit and 64 bit, and for different number of concurrent client requests. The "Cache miss count" and context switches are measured; two derived parameters namely Core Configuration effectiveness (CCE) and Intra Process efficiency (IPE) are defined to understand the performance of the application for various core configurations.

Phase II deals with two questions specific to client server configurations:

1. ability to use the kernel level parameters identified in Phase I, and observe the behaviour of the application at the kernel level and
2. ability to quantify and relate the kernel behaviour [231] to the application behaviour.

There are three test cases in Phase II that represent the spatial and temporal characteristics in client-server configuration.

Phase III focuses on hosting the web service based enterprise applications and its configuration and performance on Multi-core servers. The questions related to Connection-Orientation and Infrastructure Awareness are elicited in this phase. The integrated knowledge of application designer and the system administrator about the infrastructure is used to arrive at the optimal deployment configuration.

Phase IV deals with the methodology of arriving at threshold values for deployment configuration parameters that further establishes the concepts of Infrastructure Awareness and Connection-Orientation. The feedback injection points are identified from the overall system level; the associated impact is measured using "number of Cache misses" and the efficiency of the feedback is presented as Intra Process Efficiency. This phase also deals with performance improvements due to deployment specific contextual dispatch of requests due to Connection-Orientation.

The test bed setup of Phase V is the same as Phase IV. Phase V is set up to elucidate the effect of memory subsystem on the Multi-core behaviour. This phase deals with two questions, *viz.* relevance of data nearness and temporal locality characteristics that are specific to enterprise applications. The phase establishes usage of COAF - by using the knowledge of the memory alignment to improve the performance of the overall system.

5.1 SETTING UP TEST ENVIRONMENT

A summarised overview of the various test-bed configurations from Phase I to Phase V is presented below. This set up identifies the common platforms both hardware and software across the phases, while highlighting the differences inter-phase.

5.1.1 Generic set up - Hardware and Operating system

1. Hardware Server: Sun Niagara T2000 server with 4 CPUs and each CPU with 8 cores inside, thus totally 32 cores in a single server. This server is configured for various core combinations using Solaris Zones and container setup, such as 1 core, 2 cores, 4 cores, 8 cores etc. 32 GB DDR2 RAM, and 146 GB Hard disk.
2. CPU: UltraSPARC T1 processor with SPARC V9 architecture , 16 K instruction cache, 8K of data cache, 3MB of integrated L2 cache.
3. Server Operating System: Solaris 10 OS 11/06

5.1.2 Generic set up - Application Software Stacks

5.1.2.1 Software Versions:

1. MySQL Server v5.0.33 [130] [172], both 32 bit and 64 bit configuration.
2. Web services XAMPP [250] stack - Apache, MySQL, PHP and Python.
3. COAF module stack - Apache Axis2 [9], Apache Muse [15], NaradaBrokering, Excel services [160], DTrace [103]. - 15 DTrace probes were used as is or modified from Brendan Gregg's tool kit. The details of these *probes and the modifications* required to address the needs of experiments can be found in Appendix VI.

5.1.3 Generic Set up - Choice of Enterprise Applications

Cognizant Technology Solutions [51], Fortune 500 Company and 700 of its customers, many of them Fortune 500 companies, thus represents a good sample set for enterprises. Cognizant develops varied Enterprise Applications developed by for its customers, including itself. These applications are excellent representatives of standard enterprise applications. Over 200 enterprise use cases implemented by Cognizant Technology Solutions were studied, from both product implementations perspective (like SAP [195], [196] [197], and [230]) as well as custom implementation perspective.

5.1.3.1 Generic Enterprise Application configurations

These applications were deployed in one of the four configurations as shown in Figure 5-2 below. The four configurations are:

1. Native client calling the server where client and server are running in same deployment environment (Phase I deployment).
2. Native client calling the server where Client and server are running in different deployment environment (Phase II deployment).
3. Web service client calling the server where client and server are running in different deployment environment (Phase III and Phase IV deployments).
4. Web service client calling the server where web service client and server in same deployment environment (Phase V deployment).

5.1.3.2 Generic Enterprise Application deployment

The application is deployed with the client and server separated by a firewall as shown in Figure 5-3 with the Niagara based machine as the server. The deployment setup specific for each phase is detailed in the sections specific to each phase. All software and tools mentioned here are installed and configured using standard processes defined by respective vendor as mentioned in the reference manuals.

5.1.3.3 Phase specific - Software setup:

1. Niagara Server Stack
    a. MySQL server (Phase I and Phase II) – The MSQL Native client calls the MySQL daemon, which in turn requests the MySQL server for processing.
    b. MySQL server along with web services XAMPP stack (Phase III)
    c. MySQL server, web services XAMPP stack and COAF module stack (Phase IV, and Phase V)
2. Software client:
    a. MySQL terminal services client installed along with Niagara server (Phase I)
    b. Telnet client installed at Intel workstation (Phase II)
    c. Web services client, a PHP application installed at Intel workstation (Phase III, Phase IV and Phase V).

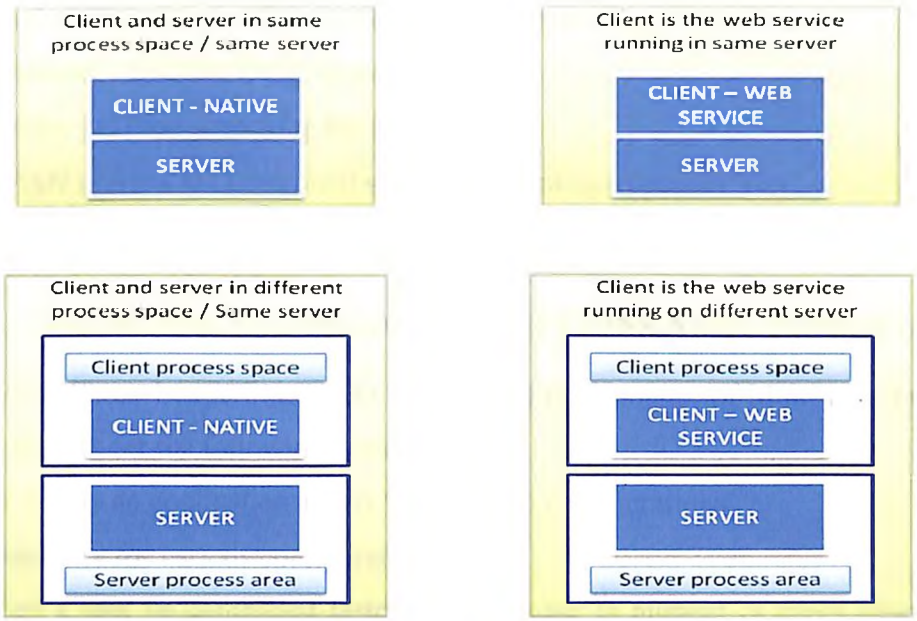


Figure 5-2 Four deployment Configurations for the test bed

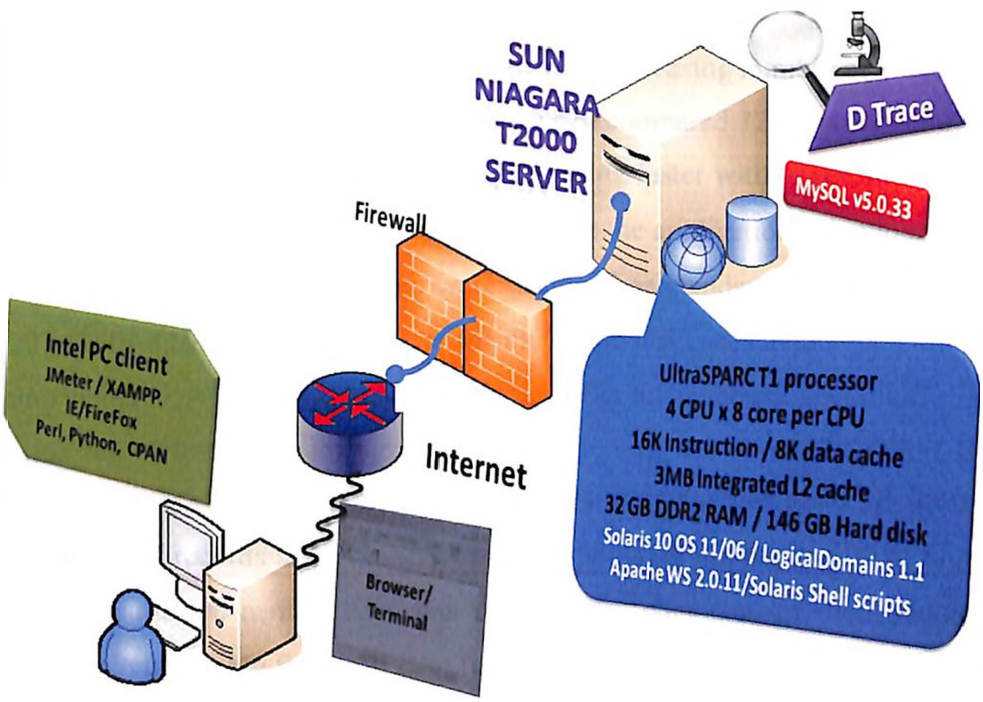


Figure 5-3 Generic Set up of the Enterprise Test bed for all applications

3. Additional software tools
    a. Logical Domains 1.1 (LDOMS) for core partitioning [174]
    b. Browsers - Internet Explorer and Firefox
    c. JMeter [13] for generating the load
    d. CPAN utilities [53] and Shell scripts (detailed in Appendix VII)

The setup is also validated for the standard blueprint test cases [173] [171].

5.2 PHASE I – VALIDATE THE BASIC ASSUMPTIONS ABOUT MULTI-CORE

The objective of Phase I experiment is to validate the performance of Multi-core to process the enterprise workloads for the following questions:

- ability to deploy an application across various core configurations
- linear scalability for various configurations
    - Can a task be completed faster with increase in number of cores allocated for processing that task?
    - Can a complex task completed faster when compared to simpler task with increasing number of cores?
    - Can groups of tasks be completed faster with increasing number of cores?
    - What is the impact of 64 bit processing when compared 32 bit processing?
    - For the same number of cores, is the processing faster with larger main memory?
- kernel parameters that need to be identified to reflect the application behaviour

To answer the above questions, test cases were designed as standardized in literature [37], [85], [105], [120], [150], [259] for phase I experiments. These test cases depict the performance of the application under varying workloads as well as different database operations. The test cases are executed for various core configurations namely, single core configuration, 2 cores configuration, 4 cores configuration and 8 cores configuration. The test cases are designed to get the specific number of records as result sets from the data base application. The number of records returned as results are namely one data record, fifty data records, five hundred data records, thousand data records, ten thousand data records and twenty thousand data records. For example, a test case for returning fifty thousand records as a result set is: "Read query: select * from contacts where id <= 50;" and a test case for returning twenty thousand records as a result set is: "Read query: select * from contacts where id <= 20000". These Test cases are also modelled as

SQL queries in two sets. The first set is based on the operations viz. Create, Read, Update and Delete. The second set of test cases deal with increasing complexity of query to determine the performance and power of Multi-core to execute either complex tasks or various volumes.

5.2.1 Phase I - First Set of test cases – Create Read Update and Delete

The first set of four tests involves queries that perform CRUD (Create, Read, Update and Delete) operations of the database application. The data structure for the records is presented in Table 5-2. Table 5-3 represents the CRUD operations that are executed over the 10 million records in the database.

5.2.2 Phase I - Number of cores to system performance is not consistently linear

Figure 5-4 consolidates the results of the first set of experiments. The results express the time scale in seconds for each of the tests CRUD tests that were run for every core configuration. The following observations can be made from Figure 5-4. There is no perceivable difference in the performance of the application due to increasing core up to 1000 records. This behaviour implies that L2 cache is not yet saturated. Beyond 1000 records, the following differences in the behaviour are observed. There is a consistent behaviour for increasing number of records for each of the operations. Also, the performance of a single core is of the same order to the performance of any of the Multi-core configurations. This defeats the linear scalability that is expected from Multi-core and hence needs to be investigated further in the second set of test cases for Phase I.

5.2.3 Phase I – Query complexity and the data size affects the performance

The second set of test cases deal with increasing complexity of query. The data structure for the second set shown in Table 5-4. This record set is used for executing the six queries – *viz.* Simple Select, Complex Select, Simple Where, Complex Where, Simple Select & Simple Where and lastly, a Complex Select & Complex Where - as represented in Table 5-5.

5.2.4 Phase I – Performance due to architecture difference is marginal

Tests are conducted for 32 bit and 64 bit architectures. This is to understand the behaviour due to variation in the architecture. For the same architecture, different data sizes are tried out. The results of the of these tests for each of the six queries, across the 32 bit architecture for both 1000 and 10000 records are shown in the Figure 5-5, Figure 5-6; similarly the results for 64 bit

architecture for both 1000 and 10000 records are shown in Figure 5-7 and Figure 5-8 respectively; the results are consolidated in Table 5-6. From the Figure 5-5 and Figure 5-6 for the 32 bit architecture, the following observations can be made:

1. The response time increases as the complexity of the query increases.
2. Adding more cores does not give the desired linear scalability. This is consistent across all core configurations for both 1000 records and 10000 records across all the six queries.

For 64 bit architectures, when the results are compared across Figure 5-7 and Figure 5-8; the three results are similar or marginally different when compared with those of the 32 bit *i.e.* (1) response time increase (2) adding more cores for linear scalability and (3) linear increase from 1000 to 10000. Inter-Architecture results

The comparison of results across architecture systems viz. 32 bit architecture versus 64 bit architecture yields the following observations.

1. The results of performance for 32-bit 1000 records (Figure 5-5) and 64-bit 1000 records (Figure 5-7) are similar and therefore, migration from 32 bit to 64 bit architecture for 1000 records does not offer any significant advantage.
2. Similarly for 10000 records as shown in Figure 5-6 and Figure 5-8, there is no significant variance or advantage in moving from 32 bit to 64 bit architectures.

5.2.5 Phase I – Complex query performs better in multi-core configuration

Apart from the linearity of response time, the load is varied by increasing the number of concurrent users from 50 to 500. This performance is captured for both 32 as well as 64 bit architectures. Figure 5-9 presents the performance of 500 concurrent users, for three of the complex queries across 1, 2, 4 and 8 cores configurations. The following observations can be made for the results.

- As the work load and its complexity increases, adding more the number of cores gives better performance.
- The performance ratio of eight cores to four cores is not linearly equivalent when compared with the performance ration of four cores to two cores. Although there is scalability for increasing number of cores to process the work load, it is not linear.

Table 5-2 User profile details in the MySQL database

User profile details	
Id	bigint(20)
Name	varchar(30)
Salary	bigint(20)

Table 5-3 First Set of Queries – CRUD Test cases

Query type	Query description
Create	LOAD DATA LOCAL INFILE '/var/home/138742/to.sql' INTO TABLE contacts;
Read	select * from contacts where id <= 20000;
Update	update contacts set name = 'f' where id <= 20000;
Delete	delete from contacts where id > 9980000;

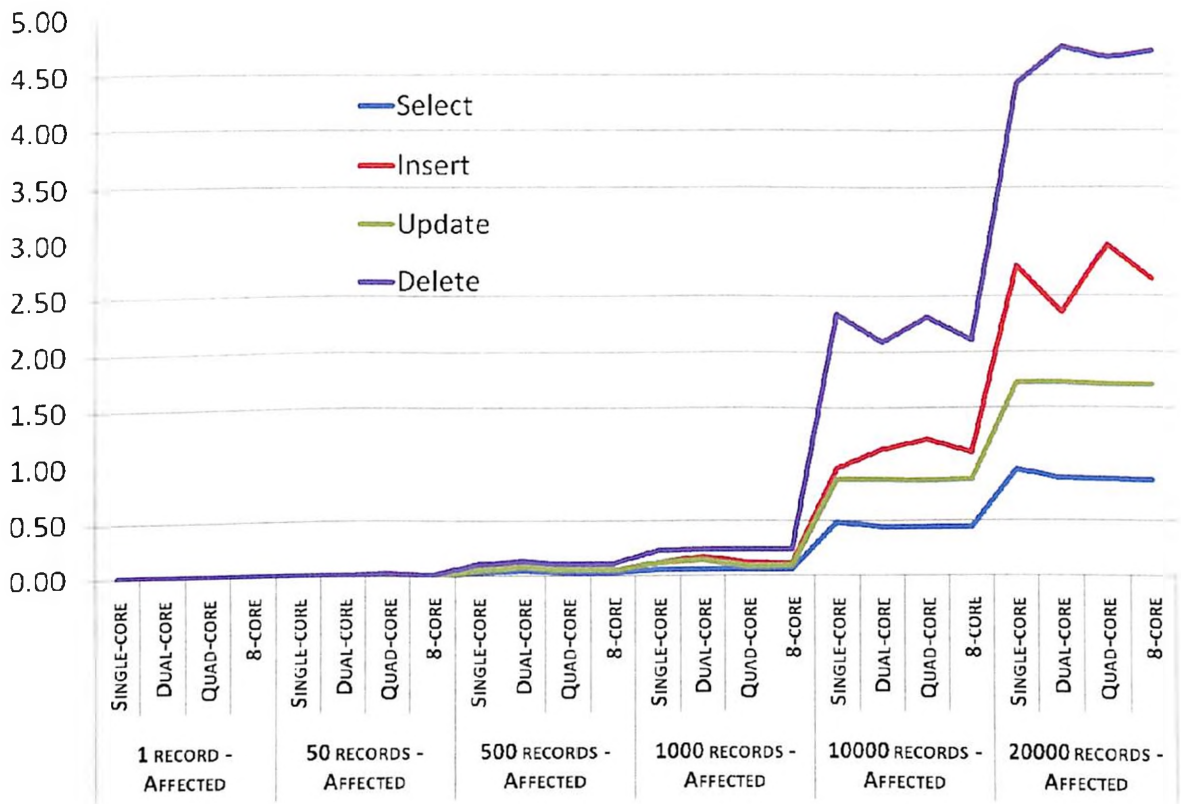


Figure 5-4 Performance of Select Insert Update Delete operations

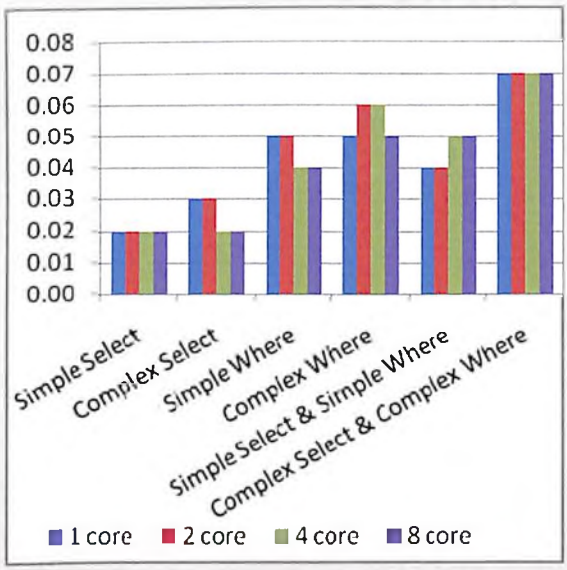
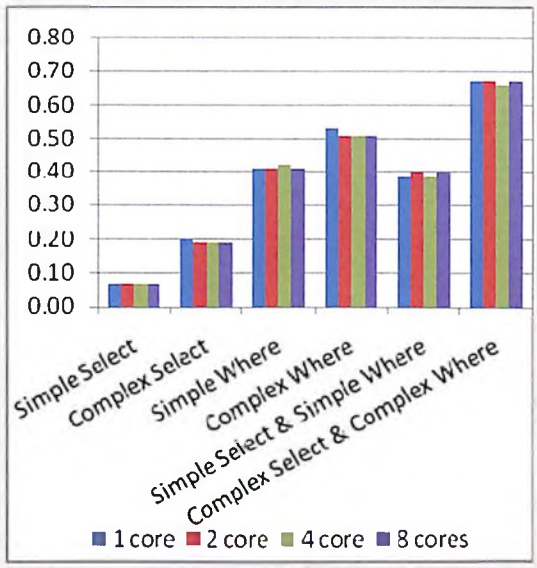
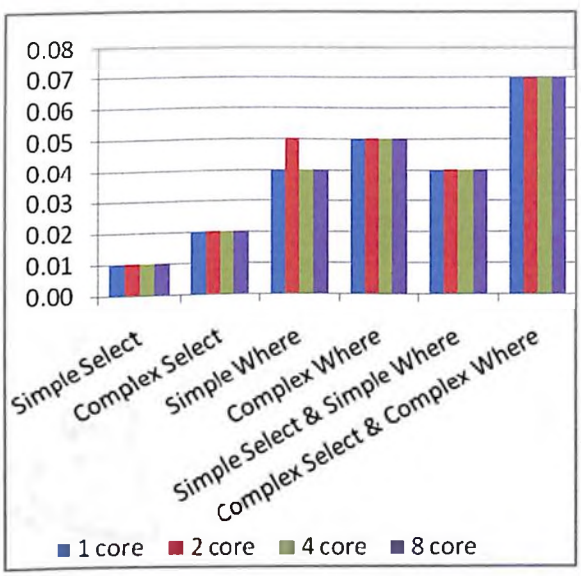
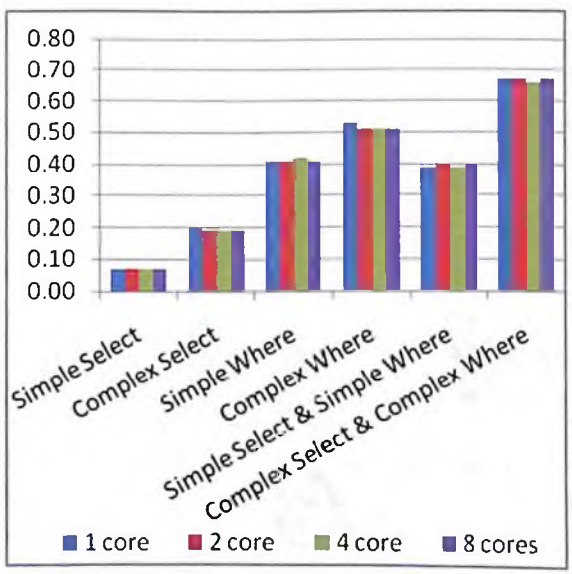
Table 5-4 Employee Details Table in the MySQL database

Employee_details	
Id	bigint(20)
Name	varchar(30)
join_date	date
designation	varchar(30)
gender	varchar(5)
salary	bigint(20)

Table 5-5 Set of Queries for the simple and complex tests

Query type	Query description
Simple Select	`select concat(substring(name, 2, 3), '-', substring(name, 3)) as result from employee_details limit 0, 1000;`
Complex Select	`select concat(case gender when 'M' then 'Mr' when 'F' then 'Ms' end, upper(name), ' has joined in ', monthname(join_date)) as result1, concat(case gender when 'M' then 'he' when 'F' then 'She' end , ' is working as ', designation) as result2, concat(case gender when 'M' then 'His' when 'F' then 'Her' end , ' id is ', substring(name, 1, 1), year(join_date)) as result3 from employee_details limit 0, 1000 ;`
Simple Where	`select * from employee_details where id between 4476000 and 4476999 and name like 's%' ;`
Complex Where	`select * from employee_details where (id between 4476000 and 4476999) and (length(salary)>5 or DATE sub(curdate(), INTERVAL 5 Year) > join date or designation like 'PM' ) ;`
Simple Select & Simple Where	`select concat(substring(name, 2, 3), '-', substring(name, 3)) as result from employee_details where name like('s%') and (id between 4476000 and 4476999);`
Complex Select & Complex Where	`Select concat(case gender when 'M' then 'Mr' when 'F' then 'Ms' end , upper(name), ' has joined in ', monthname(join_date)) as result1, concat(case gender when 'M' then 'he' when 'F' then 'She' end , ' is working as ', designation) as result2, concat(case gender when 'M' then 'His' when 'F' then 'Her' end , ' id is ', substring(name, 1, 1), year(join_date)) as result3 from employee_details where (id between 4476000 and 4476999) and (length(salary)>5 or DATE sub(curdate(), INTERVAL 5 Year) > join date or designation like 'PM') ;`

Table 5-6 Employee Details Table in the MySQL database

Arch	1000 Records	10000 records
32 –bit		
	Figure 5-5 Performance record - 32 bit - 1000 records	Figure 5-6 Performance record - 32 bit - 10000 records
64-bit		
	Figure 5-7 Performance record - 64 bit - 1000 records	Figure 5-8 Performance record - 64 bit - 10000 records

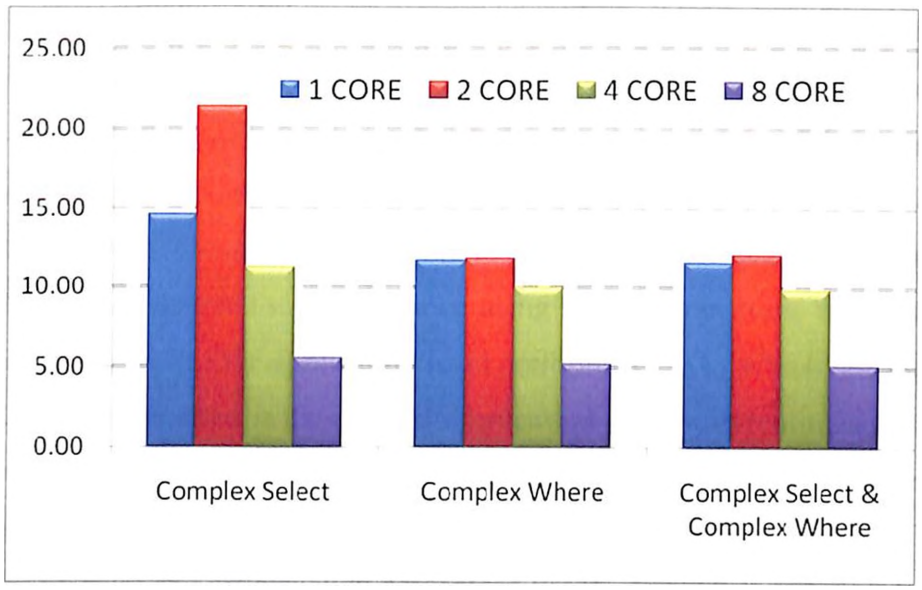


Figure 5-9 Performance 32 bit for 500 concurrent users

- As the complexity of the query increases, increasing the number of cores results in better performance.

5.2.6 Phase I – Overall summary

In summary, the user perceived performance results in Phase I shows that

- For a single application request, by varying the number of records in the result sets do not produce any pronounced effect on the performance of the overall system.
- The performance is marginally different in the case of 32 or 64 bit cores.
- Enterprise class system performs poorly on Multi-core with the application workloads lower than "ideal loads".

Hence, there is a need to investigate at the operating system level, to see how the operating system schedules a particular task and how efficiently this task gets executed. This might also gives reasons why the system is not linearly scaling. It is also imperative to identify metrics that can validate the performance in Multi-core. The effect of increasing load and load patterns also needs to be investigated.

5.3 PHASE II TO PHASE V - CONSISTENTLY MEASURE PERFORMANCE

The objective of experiments from Phase II to Phase V is to correlate the performance of kernel level parameters to the performance of the application. Observations made at *the kernel level* are used to describe the performance of the application, irrespective of the underlying core configuration that is used for processing. Three standard reference queries are run across Phase II to Phase V. The difference amongst these phases is the way the test bed is configured in each phase. To consistently determine the performance using kernel parameters, the following three questions are used to determine the application level performance. (1) Which process is running on a specific core? (2) How **often** is the **desired application** process executing/scheduled on the core? and (3) Is the performance of the **desired application** calls **efficient**, while executing in that core? To answer the above questions, two parameters are proposed namely CCE and IPE to represent the effectiveness and efficiency in percentage [234]. These two parameters are derived from the inputs gathered by observing the kernel. These two parameters could in turn, represent the relative performance of the application consistently on the Multi-core. The formulation of CCE and IPE assumes relative indication of desired process occupying the core and how efficiently it executes during that time. Additionally, the formulation assumes equal weightage of one, for all participating variables in the equation.

5.3.1 Core Configuration Effectiveness (CCE)

The Core Configuration Effectiveness (CCE) is about occupancy of the desired process in the core. It is the first parameter, which refers to how much percent of the time "a desired application process" runs when compared with "other application processes". This is based on the scheduling effectiveness of the operating system and is determined by the number of schedules that a process gets in a specified time slot.

CCE is the ratio of the number of desired application process run as a fraction of the all the other processes, (the desired process + operating system core processes + other application processes not related to the main application). Operating system processes are not considered for effectiveness, as they are required anyway irrespective of whichever application we execute on the system. Thus,

$$CCE = \frac{100*(APP_{runs})}{(APP_{runs})+(NonAPP_{runs})}$$

APP_{runs} = Sum of Count of the relevant application related processes (for example all SQL

processes)

$NonAPP_{runs}$ = Sum of Count of the all processes belonging to other applications (for example tail, DTrace processes etc.), but not related to main application in focus.

For the most ideal condition (let us say CCE as 100%), all the processes that are scheduled in a given core is related to desired application process.

The focus of CCE is to get maximum number of schedules for relevant application when compared to all other applications in the core.

Table 5-7 illustrates an example of CCE values for a test Client-Server tests across the three query. The number of APP_{runs} and $NonAPP_{runs}$ are obtained from the DTrace probe `syscall_by_proc`. Desired application process in this example is SQL process. Each record in the `syscall_by_proc` log specifies the process and its count in the window of monitoring. It can be observed from the table, that CCE for one core configuration is 84.3%. This indicates that SQL based applications utilized 84.3 % of the processing time for executing query 1 in the single core configuration and the remaining 15.7 % of the time was utilized by executing other applications processes, e.g. telnet, tail etc. Similarly, on a 32 cores configuration, SQL processes relevant to Query 1 is executed for 13.6% of the processing time and remaining 86.4% of the time was utilized for other applications processes. Thus, CCE indicates the occupancy level of the application process in the core and the CCE value closer to 100% represents the best case (ideal) scenario and vice versa.

Table 5-7 Core Configuration Efficiency for a 320K runs of Client-Server baseline

No of Cores	Query 1	Query 2	Query 3
01	84.3	60.5	82.1
04	48.4	21.3	21.2
05	56.8	21.8	41.1
08	42.5	4.6	4.3
09	62.3	47.5	42.1
16	82.1	51.1	16.9
17	83.0	49.1	75.1
18	62.7	46.4	62.4
32	13.6	66.1	89.1

5.3.2 Intra Process Efficiency (IPE)

Intra Process Efficiency (IPE) is about efficiency. It is the second parameter that defines how efficiently a scheduled "application process" performs with reference to "memory faults/Cache misses" for that process.

$$IPE = \frac{(100 - MP) * 100}{(100 - MP) + (PerCacheMiss)}$$

where,

$$MissPenalty\ (MP) = \frac{100 * (NonAPP_{runs} + NonAPP_{misses})}{(APP_{runs} + APP_{misses} + NonAPP_{runs} + NonAPP_{misses})}$$

and,

$$PerCacheMiss = \frac{(APP_{misses}) * 100}{(TotalRuns)}$$

where,

$$TotalRuns = APP_{runs} + APP_{misses} + NonAPP_{runs} + NonAPP_{misses} + OpSys_{runs}$$

When an application is scheduled to run, there are five categories of events happen on the core. They are:

1. APP_{runs} = Number of times desired application processes are scheduled
2. APP_{misses} = Number of times desired application process cache-missed
3. $NonAPP_{runs}$ = Number of runs the other application processes are scheduled
4. $NonAPP_{misses}$ = Number of times the other application processes cache-missed
5. $OpSys_{runs}$ = Number of times the operating system processes are scheduled

The most desirable state would be to schedule maximum number of APP_{runs} event and minimize all other events. However, when the application process encounters a cache-miss then operating system schedules the other process to run on the core (due to its fair scheduling policy). When the other applications also cache-miss, then problem gets compounded. So, our intent would be to calculate the impact due to relevant application cache misses and the other application cache misses. This aspect of the formulation is represented by *MissPenalty (MP)*. *MissPenalty* is the inefficiency due to relevant application cache-misses. It is desirable to reduce the number of Cache misses specific to the desired application processes in context. It can be noted that, $OpSys_{runs}$ is removed from the focus for further computations, because optimization of operating system

overheads is not the focus of the research, and the operating system is assumed to follow fair scheduling policy for scheduling its own processes. After removing $OpSys_{runs}$ runs from the total number of runs, the next step is to exclude the inefficiencies arise due to other application processes.

The second variable *PerCacheMiss* represents the impact of Cache miss in general, and is represented as the ratio of the Cache misses to cache hit. Cache miss will make the process to wait for certain number of cycles until the next fetch happens. This ratio between Cache miss to cache hit variable value is provided as guideline from the hardware manufacturer of the Multi-core server. For example, for Niagara processor a Cache miss makes the application to wait for 200 to 300 cycles [138]. Higher this ratio, more the wait, means more loss from computing perspective. Ideal situation would be not to have the Cache miss.

For example, a scenario where the total number of runs is 320 K, the summation for these five categories is depicted as below. The Table 5-8 shows the runs for one of the scenarios whether total number of runs is 320K APP_{runs}, APP_{misses}, $NonAPP_{runs}$, $NonAPP_{misses}$ and $OpSys_{runs}$, which can be depicted as:

$$\text{Total Runs} = (APP_{runs} + APP_{misses} + OpSys_{runs} + NonAPP_{runs} + NonAPP_{misses}) = 320{,}000 - (1)$$

Table 5-8 Client-Server baseline for 320 K runs on single core.

APP_{runs}	APP_{misses}	$NonAPP_{runs}$	$NonAPP_{misses}$	$OpSys_{runs}$
4,714	60,126	875	114,702	139,589

Next step is to calculate the Miss Penalty. $OpSys_{runs}$ is removed from the focus for further computations, because optimization of operating system overheads is not the focus of the research, and the operating system is assumed to follow fair scheduling for itself. After removing $OpSys_{runs}$ run from the total number of runs, the next step is to exclude the inefficiencies arise due to Other application processes in Table 5-8.

$$MissPenalty\ (MP) = \frac{100 * (NonAPP_{runs} + NonAPP_{misses})}{(APP_{runs} + APP_{misses} + NonAPP_{runs} + NonAPP_{misses})}$$

$$MissPenalty\ (MP) = \frac{100*(875+114{,}702)}{(4{,}714+60{,}126+875+114{,}702)} = 64.06\%$$

Value of MP for the application is 64.06% for the observations shown in Table 5-8. This directly maps to other processes including Cache misses (APP_{misses}, $NonAPP_{runs}$, and $NonAPP_{misses}$)

share of 175.703 runs in 320K runs is 64.06%. Therefore desired application process share (APP_{runs}) of 4714 runs within the total number of 320 K runs is 35.94% (100 – MP).

The second variable *PerCacheMiss* represents the impact of Cache miss in general, and is represented as the ratio of the Cache misses to cache hit.

$$PerCacheMiss = \frac{(APP_{misses})*100}{(TotalRuns)}$$

$$PerCacheMiss = \frac{(60126)*100}{(320000)} = 18.79\%$$

The final step is to calculate the IPE using the MP and PerCacheMiss variables.

$$IPE = \frac{(100 - MP) * 100}{(100 - MP) + (PerCacheMiss)}$$

$$IPE = \frac{(100 - 65.06)*100}{(100 - 65.06)+(18.79)} = 65.67\%$$

In our current example, the ratio of the wasted runs is (60216/320000)*100 = 18.79%. This additional unnecessary overhead that can be minimised.

This indicates, the application execution works at 65.67% efficiency. When there are no APP_{misses} the IPE is 100%.

5.3.2.1 An example of interpreting results using CCE and IPE

Table 5-9 shows the calculations for CCE and IPE for 1, 4, 5, 8, 9, 16, 17 and 32 cores.

- 4 cores and 8 cores configuration produce good results and
- 17 cores configuration produces comparatively bad results.

In 4 cores configuration, the CCE is 36.9% which means, the operating system can schedule the application for 36.9% of the time. In other words the application can occupy the core for 36.9% of the processing time. Within this time, it can complete the task in 82.9% efficiency as IPE value 82.9% indicates. Similarly, in 8 cores configuration, the CCE is 33.3% which means, the operating system can schedule the application for 33.3% of the time. Within this time, it can complete the task in 86.7% efficiency (IPE is 86.7%). Both these are desirable deployment configurations amongst all the configurations tested.

In 17 cores configuration, the operating system can schedule the application tasks in 65.6% of the time (CCE is 65.6%), but it can execute those application tasks with 33.4% efficiency (IPE is 33.4%) only. Therefore, 17 cores configuration may not be the best configuration compared to 4 cores or 8 cores configuration.

Table 5-9 CCE for 320K runs of consolidated Client-Server baseline

No of Cores	Cache Misses for the SQL	CCE for 320K	IPE for 320K
01	2256	72.8	23.5
04	1243	36.9	82.9
05	9074	42.3	66.5
08	5610	33.3	86.7
09	9952	52.5	45.1
16	13880	73.9	44.0
17	19853	65.6	33.4
18	24909	56.3	48.1
32	24443	64.6	32.9

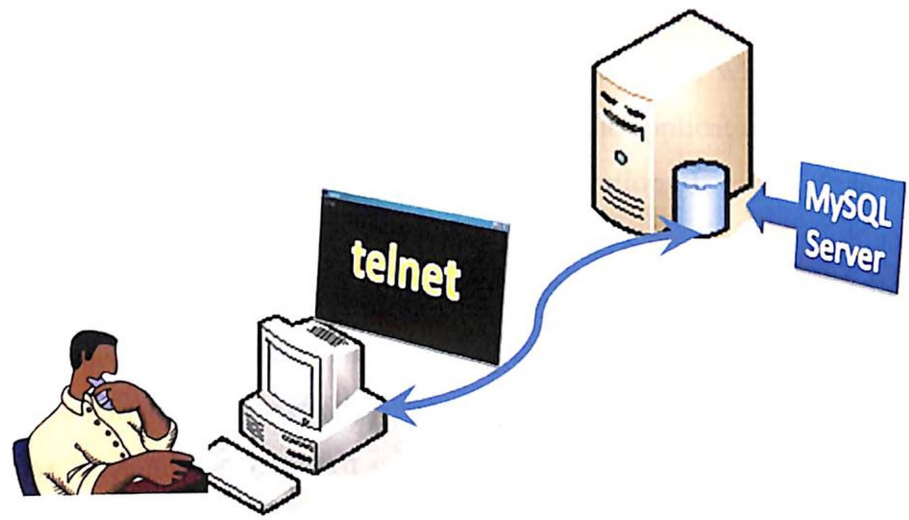


Figure 5-10 Enterprise Client-Server Application Configuration

5.4 PHASE II - CLIENT-SERVER CONFIGURATION

The objective of the Phase II experiment is to measure *kernel level* parameters and relate these parameters to application level deployment configuration parameters [[231], [233]]. The test bed is set in the standard Enterprise Client-Server mode, wherein the baselining of the system's performance is done. Three standard SQL queries commonly used in the enterprise are used as a reference. In this experiment setup, the MySQL server runs at Niagara server; client can access the server via a terminal client process as shown in Figure 5-10. The three application test case queries as shown in Table 5-10, namely Query1, Query2 and Query3 are used. These test cases are related to the typical standard cards and payments application.

- Query1 retrieves the results of the transactions for a credit card swipe that exceeded a particular limit;
- Query2 retrieves the records when the number of transactions exceeded particular count;
- Query3 retrieves the sum of all the transactions for a particular credit card.
- Query1 and Query2 represent the case for spatial locality, whereas Query3 represents both spatial and temporal locality characteristics of the application.

The test cases are run for the ten different core configurations 1, 2, 4, 5, 8, 9, 16, 17, 18, and 32 cores. The selection of these ten different configurations includes the combination of single core for base-lining against existing single core architectures, asymmetric core configuration and symmetric core configuration as shown in Table 5-11. In this phase, the MySQL parent process called "MySQL", is started when MySQL server is initialized on the Niagara server. The parent "MySQL" process is shutdown between every test to ensure the same environment setup is available before every new test case run. The DTrace logs are accumulated and analyzed to observe scheduling behaviour of the operating system. Section 5.4.1 through 5.4.6 analyzes and discusses the results of the "9 cores configuration". Similar experiments and analysis is done for other core configurations. These repetitive experiments are done to ensure and establish the consistency of the behaviour of the operating system across various core configurations. The first set of analysis is to establish the "fair-scheduling policy" of the operating system especially in the context of Multi-core systems. This helps us to validate the assumptions that the performance of an enterprise system can be improved without modifying the operating system or application binaries, as either the operating system change or modification of application binary is considered an enterprise risk as highlighted in chapter 2.0 and chapter 3.0.

Table 5-10 Three SQL query statements on an enterprise test bed

Query type	Query description
Query 1	SELECT c1.ch_cardno, date_format(c1.txn_date,'%m'), avg(c1.txn_amt) FROM ch_transaction c1 , (SELECT b.ch_cardno,count(b.txn_amt) FROM ch_transaction b GROUP BY b.ch cardno ORDER BY 2 desc limit 0,10) as t1 WHERE c1.ch cardno = t1.ch cardno GROUP BY c1.ch cardno, date format(c1.txn date,'%m') ;
Query 2	SELECT ch cardno, count(*) FROM ch transaction GROUP BY ch_cardno HAVING count(*) >100 ORDER BY 2 DESC ;
Query 3	SELECT c1.ch cardno, date format(c1.txn date,'%m%y'),count(*), sum(c1.txn_amt) FROM ch transaction c1 , (SELECT b.ch_cardno,sum(b.txn_amt) FROM ch_transaction b GROUP BY b.ch_cardno ORDER BY 2 desc limit 0,10) as t1 WHERE c1.ch_cardno = t1.ch_cardno GROUP BY c1.ch_cardno, date_format(c1.txn date,'%m%y');

Table 5-11 Test Bed Configurations and Design

Configuration type	Remarks
Single core	Replicate and baseline existing single core architectures
two cores, four cores, eight cores, sixteen cores and thirty two cores	Symmetric core layout with and without interconnect
five cores, nine cores, seventeen and eighteen cores	Forced asymmetric configuration to understand memory placement and access behaviour including cache and interconnect performance

5.4.1 Phase II – Fair Scheduling Policy validated using dispatcher queue length

The DTrace probe dispqlen_by_cpu (dispatcher queue length) is used to log the operating system schedules. Using the records present in this log, it is possible to determine the scheduling and loading behaviour of operating system as well as determine its fairness in scheduling. The methodology followed for the reconstruction of events that happened in the operating system scheduler is explained in detail using a systematic process in Appendices I to Appendix V. These appendices give an overall picture of how the operating system schedule and executed the processes across the cores in the Multi-core setup. The logs used for analysis are collected for 151 seconds in five time slots namely TS1 to TS5 as detailed in Appendix V. The DTrace logging starts in TS1 at 18:40:50, and its stops in TS5 at 18:43:21 as detailed in Table 5-12. This 151 second sequence is composed of five time slots, with each slot for logging for the first ten seconds. Figure 5-11 represents the actual scenario logs depicting how the loads are distributed on each core/CPU across the run for the first time slot of 10 seconds between 18:40:50 - 18:41:00. The load is scheduled in a queue before it is processed. The queue length on each core therefore, indicates of the spread of the load on that core. *e.g.* The queue length on the each of the CPU for the first ten second time slot *viz.* TS1 is 1954. This value of '1954' is seen across all the nine CPU cores starting from CPU0 to CPU8 indicating equal load on each of the nine cores. The same analysis is repeated across the other four ten second time slots namely TS2 to TS5. From the dispatcher queue length it becomes obvious that the load is spread more or less equally amongst across the cores. Loading behaviour is consistent across the five time slots TS2 through TS5 with loads lengths *approximately* 4454, 2563, 4320 and 1499 respectively.

5.4.2 Phase II – Process queue evenly spread across cores

The Table 5-13 displays the load distribution across the nine cores. The results in this table indicates that the load that is evenly spread across six cores (CPU 0 to CPU 5), whereas the queue is widespread in the remaining three cores (CPU 6, CPU7 and CPU8). After consolidating the multiple queues of each CPU/core the summary is presented in Table 5-14. For example, for CPU 6 the following load queue lengths {674, 252, 521, 330, 22, 93, 2, 60} are replaced with their sum (674+252+521+330+22+93+2+60 =) 1954.

Table 5-12 DTrace scheduled Time slots

Time Slot No	Start time	End Time
TS1	18:40:50	18:41:00
TS2	18:41:21	18:41:31
TS3	18:41:55	18:42:05
TS4	18:42:29	18:42:39
TS5	18:43:11	18:43:21

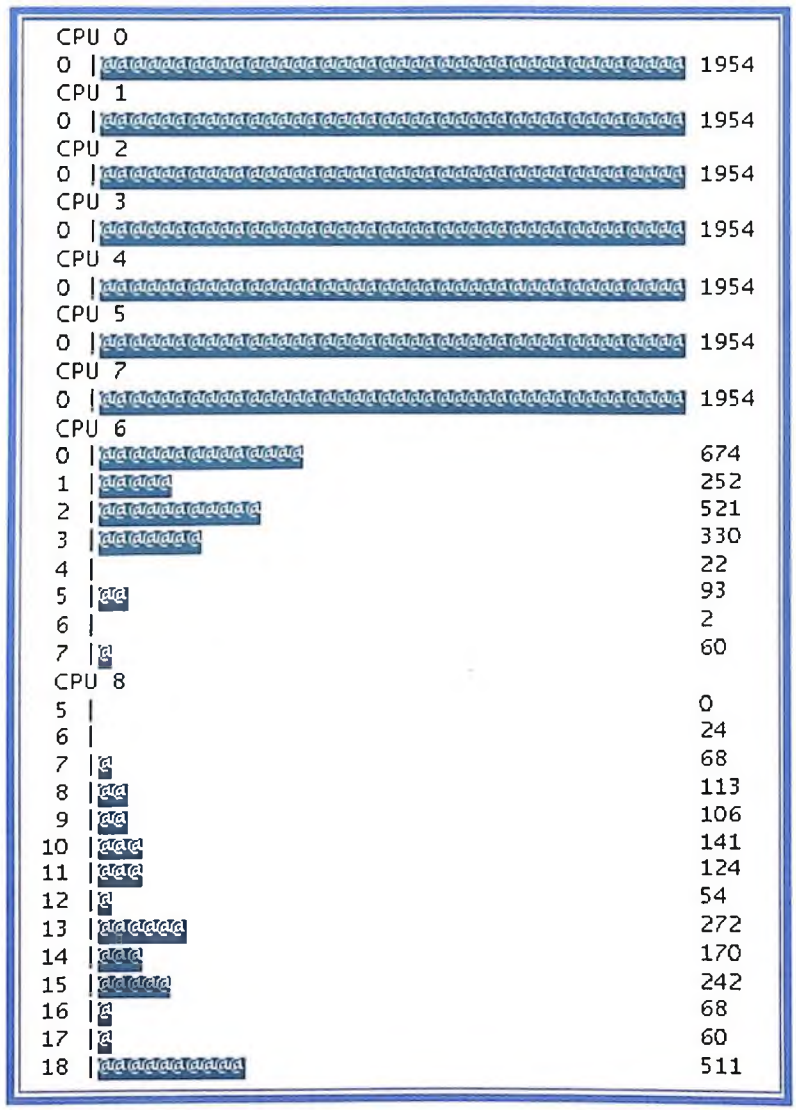


Figure 5-11 dispqlen_by_cpu as it appears in the log

Table 5-13 Dispatcher queue lengths for nine cores

CPUID	TS1	TS2	TS3	TS4	TS5	Total Schedules	CPUID	TS1	TS2	TS3	TS4	TS5	Total Schedules
CPU 0	1954	4455	2563	4320	1499	14791	**CPU 8**	24	479	27	221	161	912
CPU 1	1954	4455	2563	4320	1499	14791		68	784	338	79	13	1282
CPU 2	1954	4455	2563	4320	1499	14791		113	686	379	89	68	1335
CPU 3	1954	4455	2563	4320	1499	14791		106	289	207	11	85	698
CPU 4	1954	4455	2563	4320	1499	14791		141	159	51	8	48	407
CPU 5	1954	4455	2563	4320	1499	14791		124	227	50	60	118	579
CPU 6	674	179	626	650	1410	3539		54	277	178	120	175	804
	252	2290	414	331	70	3357		272	202	308	131	137	1050
	521	292	679	706	19	2217		170	101	150	152	37	610
	330	881	205	282		1698		242	94	136	122	49	643
	22	270	510	178		980		68	220	369	169	15	841
	93	128	10	240		471		60	256	42	378	20	756
	2	208	10	330		550		511	101	246	166	91	1115
	60	207	109	84		460		1	152	79	200	124	556
				337		337			3	4	78	202	287
				29		29			4		127	156	287
				31		31			11		15		26
				1		1			410		115		525
				430		430			1		332		333
				141		141					252		252
				305		305					150		150
				50		50					64		64
				195		195					36		36
CPU 7	1954	4214	2101	2028	1493	11790					150		150
	0	241	465	2292	6	3004					208		208
											11		11
											170		170
											211		211
											190		190
											305		305

Table 5-14 Summary of the schedules queues.

CPUID	TS1 18:40:50 18:41:00	TS2 18:41:21 18:41:31	TS3 18:41:55 18:42:05	TS4 18:42:29 18:42:39	TS5 18:43:11 18:43:21	Total Schedules
CPU 0	1954	4455	2563	4320	1499	14791
CPU 1	1954	4455	2563	4320	1499	14791
CPU 2	1954	4455	2563	4320	1499	14791
CPU 3	1954	4455	2563	4320	1499	14791
CPU 4	1954	4455	2563	4320	1499	14791
CPU 5	1954	4455	2563	4320	1499	14791
CPU 6	1954	4455	2563	4320	1499	14791
CPU 7	1954	4455	2566	4320	1499	14794
CPU 8	1954	4456	2564	4320	1499	14793

The values 1954, 4456, 2564, 4320, and 1499 in each of the five time slots is almost same for all CPU/Core, and the value of the total queue size within a small range of 14791-14794. This confirms the typical behaviour of an operating system for its fair scheduling policies through the even distribution of load across all the CPU/cores evenly.

5.4.3 Phase II – Varying loading pattern across cores

From Table 5-13 it can be observed that the loading pattern is not same across the nine cores, while the queue length (in other words the loading schedules) is almost same across them. The loads on each core are plotted for TS1 to TS5 in Table 5-15 across diagrams in Figure 5-12 to Figure 5-16 and the consolidated load on each processor is Figure 5-17. Looking into TS1 loading, the queue length pattern on cores CPU0 to CPU5 is smooth and consistent. There is only one schedule for the queue length of 1954 in CPU0 to CPU5; there are 8 schedules on CPU 6; 2 schedules on CPU7; and 14 schedules on CPU8. These results indicates the "Fair Scheduling Policy" followed by the operating system for evenly distributing the load across cores; and the difference in loading pattern is attributed to the kind of processes that are scheduled against each core. Table 5-16 indicates that CPU6, CPU7 and CPU8 ran different kinds of processes when compared to CPU0 to CPU5.

Table 5-15 Individual and Cumulative Load patterns for 9 CPU Loads

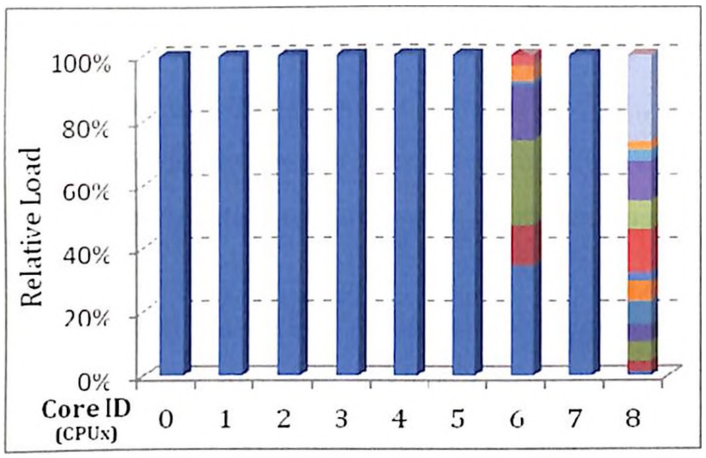


Figure 5-12 Load distribution – TS1

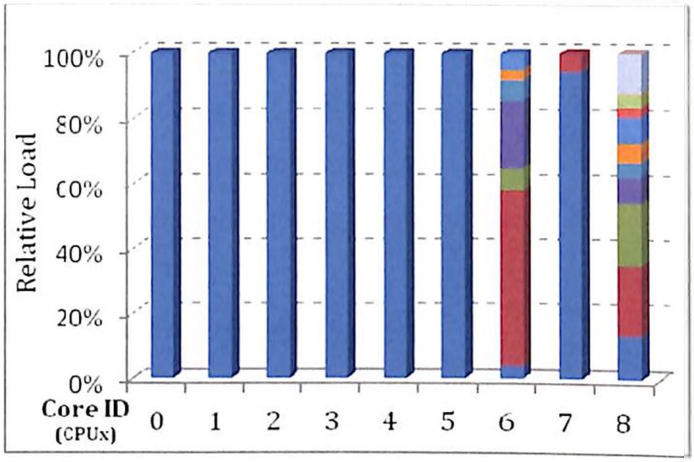


Figure 5-13 Load Distribution – TS2

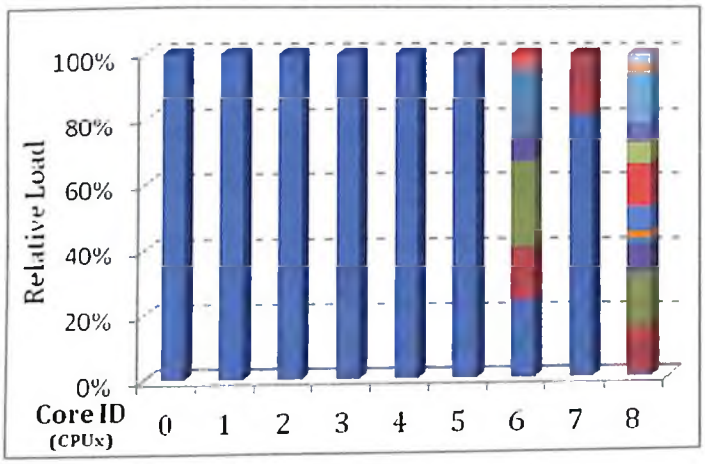


Figure 5-14 Load distribution – TS3

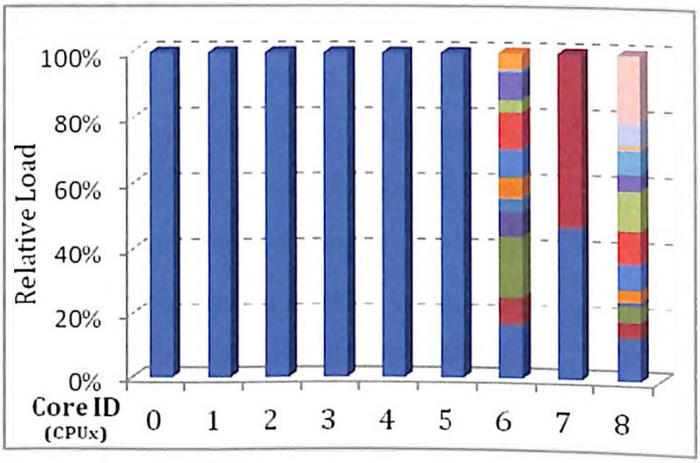


Figure 5-15 Load distribution – TS4

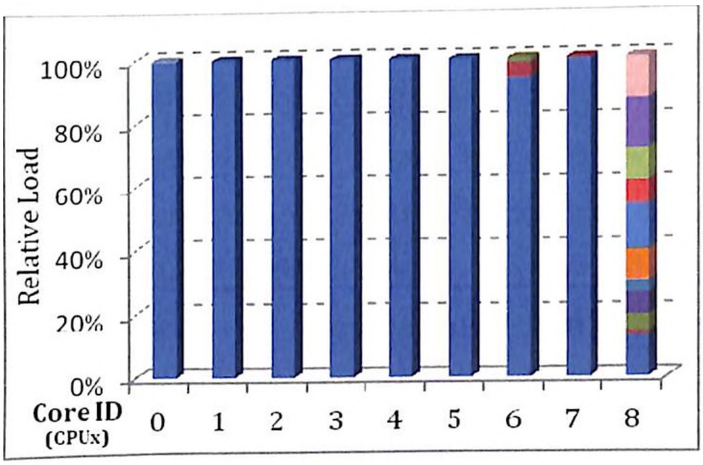


Figure 5-16 Load distribution – TS5

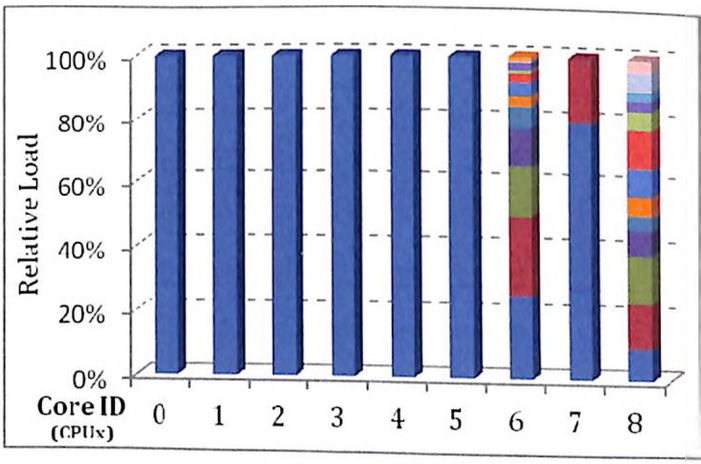


Figure 5-17 Consolidated Load distrib. TS1–TS5

The investigation using syscall_by_proc probe reveals that there are three kinds of processes scheduled on a core. These three processes are:

1. Enterprise application processes – in our current experiment it is related SQL query execution processes such as MySQL, MySQLd, t.sql, expr etc.
2. Other application processes – in our current experiment it is related to enterprise administration and management processes such as dtrace, sendmail, telnet, bash, etc.
3. System processes – in our current experiment, these are Solaris operating system processes like inetd, svcstart.d, getsockname, mprotect, mmap64, sshd etc.

The details of separating these three kinds of processes from the DTrace logs are explained in Appendix IV and V.

Table 5-16 shows the load schedule in all the five slots TS1 to TS5, for all the nine cores CPU0 to CPU8. It can be observed from this Table 5-16, CPU8 is executing the operating system processes predominantly compared to other cores in all the five time slots as depicted by the number of schedules as 14, 19, 15, 30, and 16 respectively. As observed in another view depicted in Table 5-17, CPU8 is executing the operating system processes predominantly when compared to other cores in all the time slots. This indicates that the operating system is sensible to schedule against each core instead of mixing the processes across cores arbitrarily. In time slots TS1, TS2 and TS3, operating system spends its time in organizing itself to run the MySQL and non-MySQL processes. In TS4 and TS5 it has scheduled more MySQL and non-MySQL processes compared using the time for scheduling its own processes. In summary, CPU0 to CPU5 are scheduled in ideal condition in the context of application processes. It would be ideal to have CPU6 and CPU7 scheduled similar to the scheduling of CPU0. When observed the processes in CPU6, it can be seen that CPU6 is executing other-application processes (Non MySQL processes), which could be routed to different infrastructure. Similarly, it can be observed that CPU7 is executing processes related to interconnect processes. CPU7 also could be could be scheduled to execute like CPU0, if there were no interrupts due to interconnect processes for a nine-core configuration. CPU8 is almost not available for scheduling application processes and it is needed for executing the operating system processes. Thus, we establish the opportunity for schedule improvements through Connection-Orientation for CPU6, and through Infrastructure Awareness for CPU7 on the basis of load schedule observations.

Table 5-16 Schedule loading on CPU0 to CPU8

Scheduled Queues	TS1	TS2	TS3	TS4	TS5
CPU0	1	1	1	1	1
CPU1	1	1	1	1	1
CPU2	1	1	1	1	1
CPU3	1	1	1	1	1
CPU4	1	1	1	1	1
CPU5	1	1	1	1	1
CPU6	8	8	8	17	3
CPU7	2	2	2	2	2
CPU8	14	19	15	30	16

Table 5-17 Comparison of DTrace logs on a single processor running multiple commands

Schedules for CPU8	TS1	TS2	TS3	TS4	TS5
Scheduled Queues	14	19	15	30	16
Actual SQL schedules	5	11	5	63	38
Actual nonSQL schedules	5	11	5	63	39

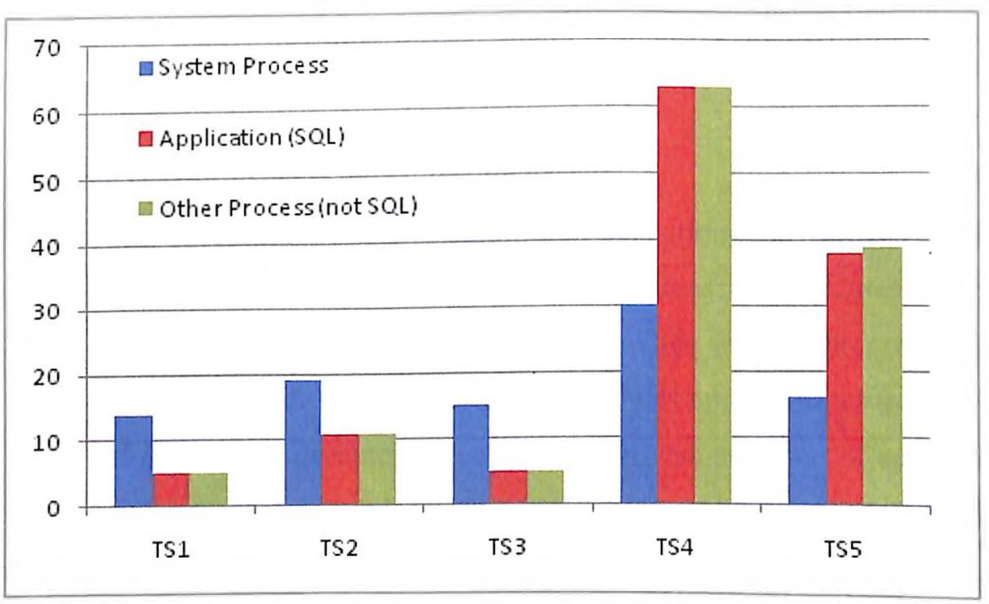


Figure 5-18 SQL vs Non SQL commands as seen by two probes

Table 5-18 shows the consolidated list of all the system calls happened in five time slots. As seen in the table, there are 83747 total system calls happened in 151 seconds across five time slots. In TS1 to TS3 application (MySQL) initialization and stabilization happened. In TS4 and TS5 the application is executing (executing SQL queries) There are 27804 calls are related to main application and 20090 calls are related to other application processes; this translates to 58% calls for main application and 42% for other applications. If we can move or reduce the number of other application calls outside the infrastructure then we can make the relevant application to occupy the core for execution. That is any reduction of 42% calls (that relates to other applications) or migration of these 42% calls to other infrastructure could improve the performance of the main application. This is in other words is the "ability to identify Connection-Orientation between the infrastructure and the application process". The detailed methodology is followed to extract, analyse and summarize the log data to arrive at the consolidated event table as depicted in Table 5-18; the methodology to extract to analyze and to summarize the logs is presented in Appendix II, Appendix IV and Appendix V respectively. This summarized view helps us to visualize and appreciate this situation occurring even for an ideal condition like in-process Client-Server scenario. In the case of executing stateless Web services the situation becomes magnified and we could see that the core is occupied by other application processes for larger duration when compared to main application process. Hence, the need for Connection-Orientation becomes a requirement to improve the performance of the web service based application.

5.4.4 Phase II – Successful process gets more focus

The next interesting observation is the scheduling of execution-ready processes in the core in a time slot. Operating system gives importance to the process that executes without cache miss, when compared to the process that misses the cache. When we plot the process schedules in a graph for a given time, we can see these process schedules appear in groups; this means a process that has all information to execute can continue to get the attention of operating system when compared to the process that do not have all information to execute. Table 5-19 details the summarized representation of "number of clusters in a graph" for all the three queries in Client-Server configuration for various core configurations (1, 2, 4, 5, 8, 9, 16, 17, 18, and 32 cores configurations); each cluster represents a process that gets scheduled and executed continuously. The cluster is obtained through a systematic statistical procedure as detailed in Appendix IX.

This cluster of "same process executing continuously", also corroborates with the large processing slots of time seen in TS1 time slot. The ideal situation would be a single process continuous to occupy the core till the completion of execution. So, less number of clusters means good execution when compared with more number of clusters. Table 5-19 shows that number of clusters increase as the number of cores increase; this could be due to increase in context switches; as the number of cores available for operating system, it starts to schedule the process across the cores; this creates a context switch which ultimately affects the overall performance of the system.

5.4.5 Phase II – Performance represented using CCE and IPE

Having obtained the number of runs and cache misses for a given core configuration, next step is arrive at the CCE and IPE values, so that the interpretation becomes easier. Table 5-20 shows the computed values for CCE and IPE for various core configurations ((1, 2, 4, 5, 8, 9, 16, 17, 18, and 32 cores configurations) in Client-Server deployment mode.

Table 5-18 TS1 to TS5 – consolidated results (as seen by syscall_by_process)

Serial No Start Proc	Serial No End Proc	Start Time of the Run	End Time of the Run	All syscalls start	All syscalls End	Total Runs of syscalls	SQL runs	NonSQL runs	SQL syscalls	NonSQL syscalls
1	112	18:40:50	18:41:00	1	3061	3061	5	5	13	856
113	615	18:41:21	18:41:31	3062	15479	12418	11	11	32	3753
616	1237	18:41:55	18:42:05	15480	23862	8383	5	5	5	12839
1238	2516	18:42:29	18:42:39	23863	41170	17308	63	63	6892	11463
2517	5209	18:43:11	18:43:21	41171	83747	42577	38	39	27804	20090

Table 5-19 Number of "Clusters" or "large sequence of executions" for each configuration

Number of Cores/ Query ID	1	4	5	8	9	16	17	18	32
q1	2	5	6	8	11	12	20	22	30
q2	2	5	8	8	14	10	16	19	24
q3	2	5	16	11	16	10	34	34	24

Table 5-20 Phase II - Client-Server - Core Config Effectiveness & Intra Process Efficiency

No of Cores	QueryID / Threads	Clusters	No of Command executions		No of Cache Misses		CCE	IPE
			SQL	NonSQL	SQL	NonSQL		
01	q1	2	4,714	875	60,126	114,702	84.3	65.7
01	q2	2	5,351	3,496	177,326	519,098	60.5	31.9
01	q3	2	4,010	875	71,114	126,634	82.1	62.5
04	q1	5	896	954	9,357	30,643	48.4	89.3
04	q2	5	151	559	3,942	14,404	21.3	94.6
04	q3	5	136	507	1,397	8,515	21.2	97.1
05	q1	6	1,438	1,094	15,935	52,791	56.8	83.0
05	q2	8	364	1,303	8,315	33,508	21.8	88.5
05	q3	16	880	1,262	13,160	42,678	41.1	85.5
08	q1	8	852	1,155	8,064	33,449	42.5	89.0
08	q2	8	11	227	178	3,920	4.6	98.7
08	q3	11	17	382	254	5,847	4.3	98.1
09	q1	11	3,244	1,966	41,449	82,416	62.3	72.8
09	q2	14	1,614	1,787	52,417	117,436	47.5	65.6
09	q3	16	1,352	1,856	21,518	83,677	42.1	75.8
16	q1	12	8,444	1,843	111,256	169,060	82.1	54.2
16	q2	10	664	635	26,039	57,085	51.1	79.5
16	q3	10	163	802	39,121	13,383	16.9	85.7
17	q1	20	3,207	657	42,037	54,898	83.0	77.4
17	q2	16	3,315	3,438	108,511	283,106	49.1	45.3
17	q3	34	3,475	1,150	57,178	107,310	75.1	66.7
18	q1	22	2,563	1,524	28,083	20,010	62.7	87.0
18	q2	19	2,584	2,981	73,355	133,539	46.4	60.9
18	q3	34	3,025	1,826	45,783	48,455	62.4	77.5
32	q1	30	272	1,730	50,020	40,575	13.6	77.7
32	q2	24	2,926	1,499	101,732	257,258	66.1	47.5
32	q3	24	3,478	426	61,516	150,818	89.1	61.0

Core Configuration Effectiveness

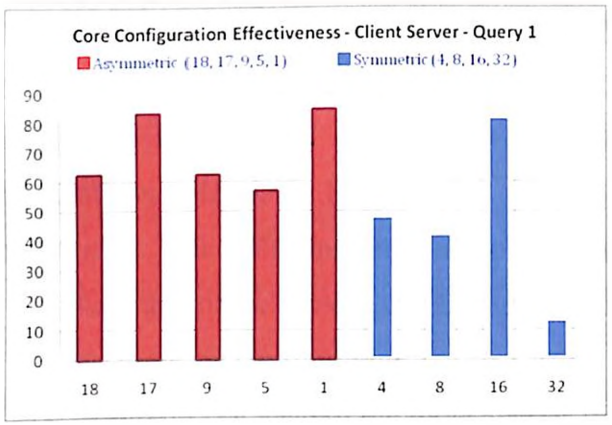


Figure 5-19 CCE for Query 1

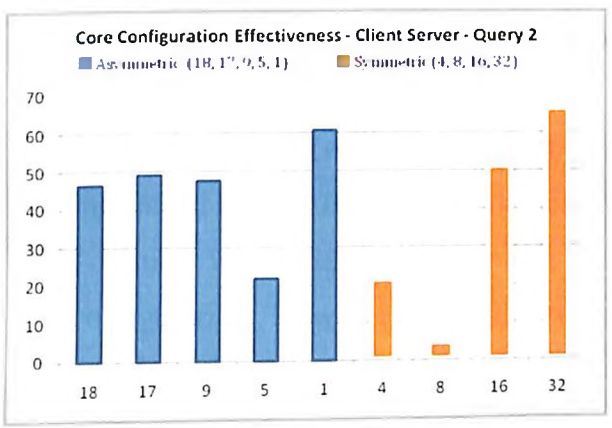


Figure 5-21 CCE for Query 2

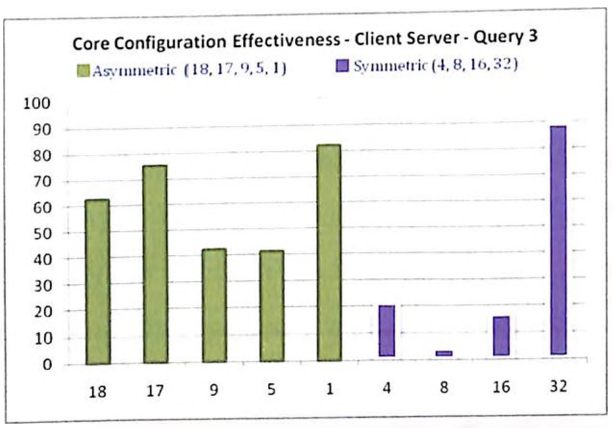


Figure 5-23 CCE for Query 3

Intra Process Efficiency

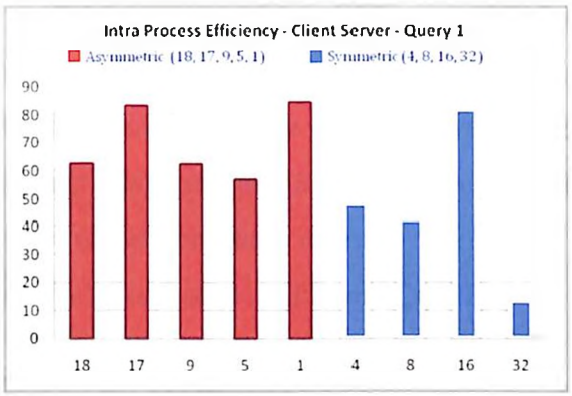


Figure 5-20 IPE for Query 1

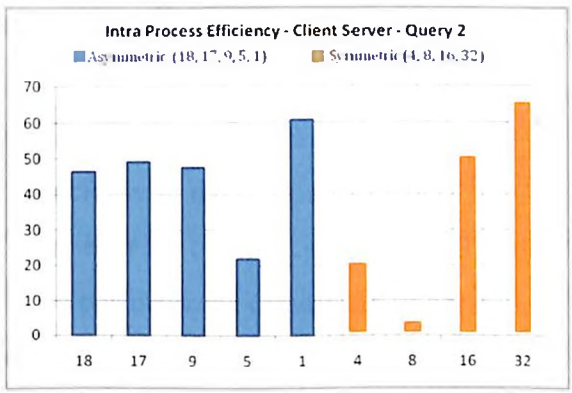


Figure 5-22 IPE for Query 2

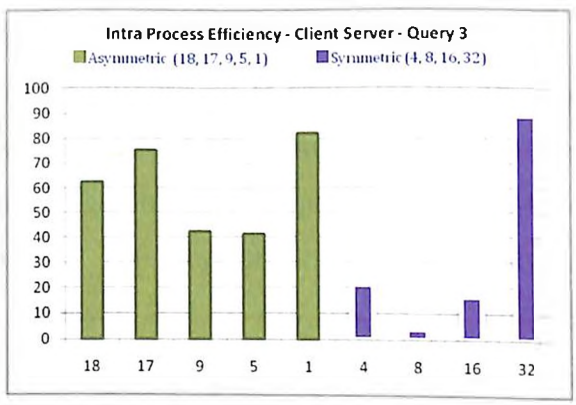


Figure 5-24 IPE for Query 3

5.4.6 Phase II – Performance for Symmetric and Symmetric Core configurations

Figure 5-19 to Figure 5-24 depict the results of the three runs Query1, Query2 and Query3 in the Client-Server mode (spatio-temporally aligned mode), represented for CCE and IPE for various core configurations. These results are represented in decreasing order of asymmetric configuration (viz. 18, 17, 9, 5 and 1) followed by a set of increasing order of symmetric configurations (viz. 4, 8, 16 and 32). This representation is done to illustrate the effect of symmetricity. The following important observations were made from these results.

1) For each of the three queries, there is a strong correlation between IPE and CCE for asymmetric configuration.
2) Similarly, for each of the three queries there is a strong correlation between IPE and CCE for symmetric configuration.
3) Another observation is that asymmetric processors perform poorly when compared to symmetric processor.
4) Increasing number of clusters against the increasing number of cores, resulting in increasing number of context switches, thereby decrease in overall performance.
5) For query wise results, it appears IPE and CCE are best for complex queries for 32 cores configuration. However, when observing per core performance (assuming linear scalability) ideally 32 cores configuration could give eight times (800 percent) more performance when compared to a 4 cores configuration. However, as seen later in Figure 6-3, the per-core performance is 1.3 (130 percent) for 32 cores processor instead of 800 percent in an ideal linearly scaling scenario.

In summary, in this Phase, we have used nine-core configuration to showcase the process the DTrace logs and the obtained results of Client-Server based deployment; we identified the opportunity to improve the performance of the system using two key concepts of COAF namely Connection-Orientation and Infrastructure Awareness. This log processing and analysis methodology is repeated for all core configurations namely 1, 2, 4, 5, 8, 9, 16, 17, 18, and 32 cores configurations; the results obtained from this phase can be subsequently used for future phases for comparing the behaviour of web services based deployment with client server based deployment.

In the Client-Server configuration, the core MySQL processes execute for maximum duration of the time with a small amount of failures when compared to non-MySQL processes

and operating system processes. This observation validates the behaviour of the application in spatial and temporal coherence, and thus exhibits the Connection-Orientation behaviour of Client-Server configuration. This behaviour is the desirable behaviour for the enterprise and our aim is to replicate the performance of Client-Server for web services for various Multi-core server configurations.

5.5 GENERIC WEB SERVICES SETUP FOR PHASES III TO PHASE V

To evaluate the Multi-core performance, we built the stack of standard COAF infrastructure components that are listed in chapter 4.0 such as Apache, NaradaBrokering, DTrace, Axis2, PHP run time, Python run time, script run times, MySQL database server, and custom components such as configurators for threads, get/set operators for threshold values, web services clients for start / stop web services, loads pumps etc. using PHP/Python. The typical enterprise application consists of multiple servers clearly separated into web servers, application servers and databases with a configuration as seen in the Figure 5-25. The Niagara server was configured to run the MySQL database server, and the data services are exposed through a web service. The database is setup with 10 million business records to reflect the nature and size of real-time enterprise data with variations in query complexity, the request arrival and volume of the result sets that are returned for those queries. The data services return the relevant result sets to the requesting client component. Apache JMeter is used for pumping the load simultaneously with various frequencies as required in the "number of threads" configurations to emulate an enterprise load setting.

5.6 PHASE III - TEST BED SET UP FOR WEB SERVICES WITHOUT COAF

The objective of Phase III is to identify the Application Design Level parameters for a web service based application and its infrastructure, establish the existing baseline behaviour for web services in the current enterprise configurations, and identify the application run-time parameters that can possibly help and refine the application level performance. In phase III, the client server application of Phase II is replaced by its web service counterpart. The default application that ran MySQL as a client is now run with MySQL as a web service along with Apache web server and app server. In Phase III, the logs of the three query results are averaged out, so that the results are focussed on inter-core performance. Also, the operating system calls are isolated from

SQL and No-SQL calls, so that experiment focus is on processes relevant to the application. Hence, the results are depicted only for SQL and No-SQL leaving out System calls. To establish the effectiveness of Connection-Orientation and Infrastructure Awareness aspects of COAF, the phase three is focused on collecting the baseline results for web services without COAF. Figure 5-26 represents the enterprise setup but the client application demonstrating the multi-user access. WAMP (Windows Apache MySQL PHP) stack is installed on the Intel based Desktop Machine. This desktop is used to pump multiple user requests concurrently. JMeter is installed in windows machine. The JMeter is used to spawning user requests. JMeter sends request to Apache web server as configured in Figure 5-27. JMeter is configuration, an HTTP Request sampler and a corresponding listener is added. A request is traced from the origin to the end destination back to Origin, to ensure that every request is processed. However, during the experiments there are some requests that may not be processed due to various reasons such as access latency, network non-availability etc. These requests are outliers for our experiment, even though they may also appear in the standard enterprise infrastructure. To remove these outliers, the listeners are setup to provide the reports on every test case.

A test case is organized so that a web service request can be pumped in various numbers and groups. On the server side, Apache is configured to acknowledge the request and send back the response. The Niagara server hosts all the three major infrastructure components namely Apache Web server, Tomcat/Axis2 Application server and MySQL database Application. A test case is a PHP based web-service program that can randomly generate a 3 digit number, which is the used for querying the credit-card number in our test case. This web service in turn queries the MySQL database tables and returns the results wrapped as web service. As shown in Figure 5-28, the test plan added with the URL http://servername.com/phpmyadmin/load.php.

5.6.1 Phase III – Web Services without COAF – Setup

JMeter is configured to send 100 or more parallel requests. A test plan is created with run-time parameters "number of client threads" (for example $n_{Threads} = 100$), rampup period as one second and LoopCount as 1. The test is run multiple times to ensure that we get consistent results. This process is run for various values for numbers of threads, such as 200 threads, 300 threads etc.; the above process of testing is conducted for different core configurations such as 8, 9, 16, 17, and 32. More focus is given to these five different core configurations (8, 9, 16, 17,

and 32), due to high CCE and IPE values. The performance results are measured and documented for each of the thread iterations and the results are documented for further analysis.

5.6.2 Phase III – CCE and IPE for Web Services without COAF feedback

Table 5-21 is the comparative results obtained for Web services setup without COAF for various core configurations (8 cores, 9 cores, 16 cores, 17 cores and 32 cores) for values of "number of client threads" as 1000 and 2000 threads (Maximum of 1000 or 2000 simultaneous connection requests in JMeter setup); these results contain number of command executions for SQL and NonSQL processes, number of Cache misses for both SQL and NonSQL processes, and the two computations CCE and IPE for each core configuration (8 cores, 9 cores, 16 cores, 17 cores and 32 cores). As seen in Figure 5-29 and Figure 5-30, when the value of "number of client threads" increased from 1000 to 2000, there is a reduction in CCE. This indicates that lower number of desired application processes (MySQL processes) scheduled on the core, when compared to other processes (such as Non-MySQL processes). Also, IPE has reduced for increase in number of threads from 1000 to 2000 as shown in Figure 5-31and Figure 5-32 respectively.

5.6.3 Phase III – Spatio-temporal characteristics observed.

The execution results for three queries Query1, Query2, and Query 3 (Table 5-10) is obtained for both client server mode and Web services mode deployments. Table 5-20 is the summary of results obtained for client server configuration and Table 5-21 is the summary of results obtained for Web services configuration. Results of these two tables are correlated between "the numbers of cores" and "the number of Clusters" using Pearson's correlation, and these results are displayed in Table 5-22 . The resultant Pearson's Correlation Coefficient is further used for interpreting various scenarios. In the Client-Server scenario the correlation is significant (0.967 and 0.922) for simple queries; this may be due to spatially coherent characteristics of Query1 and Query2. This also corroborates with the behaviour of operating system to schedule parallel tasks in favour of spatially coherent processes. By design, Query3 is temporally incoherent. Correlation Coefficient for Query3 in Client-Server scenario is low (0.666); this means, increase in "number of cores" to execute the Query3, results in increase in number of clusters. In Web services mode, the correlation is low for all three queries (0.707) when compared to the results obtained in Client-Server mode; this result is due to spatio-

temporally incoherent nature of Web services. This facilitates the need for further enquiry into temporal specific behaviour for Web services based configuration, as detailed in Phase V.

5.7 PHASE IV - COAF BASED TEST BED SET UP – WEB SERVICES

The objective of the Phase IV experiment is to test the effect in performance changes (preferably performance improvement) due to feeding back the knowledge obtained through COAF framework. The key activity in this Phase IV is to identify the configurable run time parameters, *configure* those parameters with different values, and observe the variation in performance; This observation is stored as knowledge-map and this knowledge is injected back to system as feedback; this knowledge is in turn makes the enterprise application aware of its infrastructure configuration. Figure 5-33 depicts the set up for web services with COAF modules such as Core Scheduler module, Behaviour Information module, Threshold Configuration module, and Notification Framework module.

5.7.1 Phase IV – Application Parameters – Setup

For this test setup, an additional Python based script is developed and this script is deployed as a component of Threshold Configuration module (COAFConfigurator.py). This script runs continuously; this script has the ability to reset the system after the test case execution is completed for a given core configuration. This level of automation helps us to try out various test cases and collect results so that we could plot the results against the set objectives. The "*number of client threads*" is chosen as the application level control parameter for this setup. Figure 5-34 represents the detailed view of the workflow in the COAF based Web services setup. The COAFConfigurator component creates the XML-RPC requests that are channelled through a web server to the COAF based middle tier environment. The XML-RPC listeners are bound to a specific port. This is done to ensure that other process do not get routed inadvertently into the test zone. The XML-RPC headers also contain the routing information to the data servers that render the data so that RPC headers can be used to map and bind the server to the service. Eighteen different script files that monitor the CPU, processors, the distribution and the symmetricity were written to perform this observation.

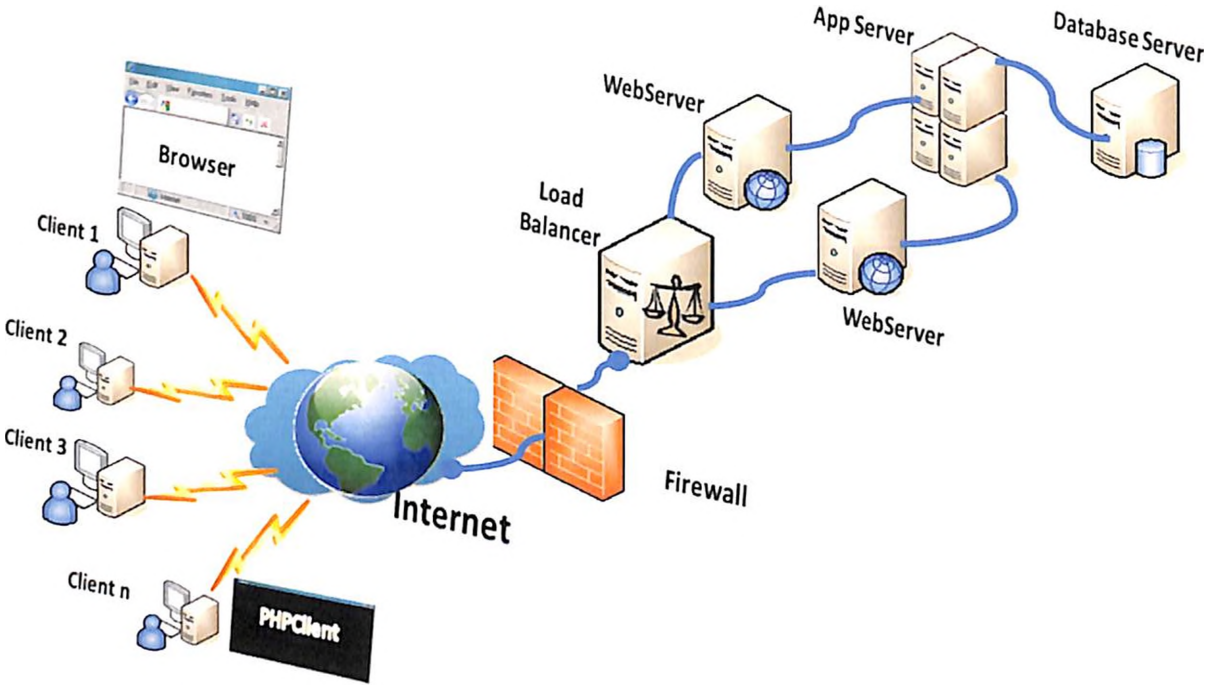


Figure 5-25 Web Service based setup for the enterprise application

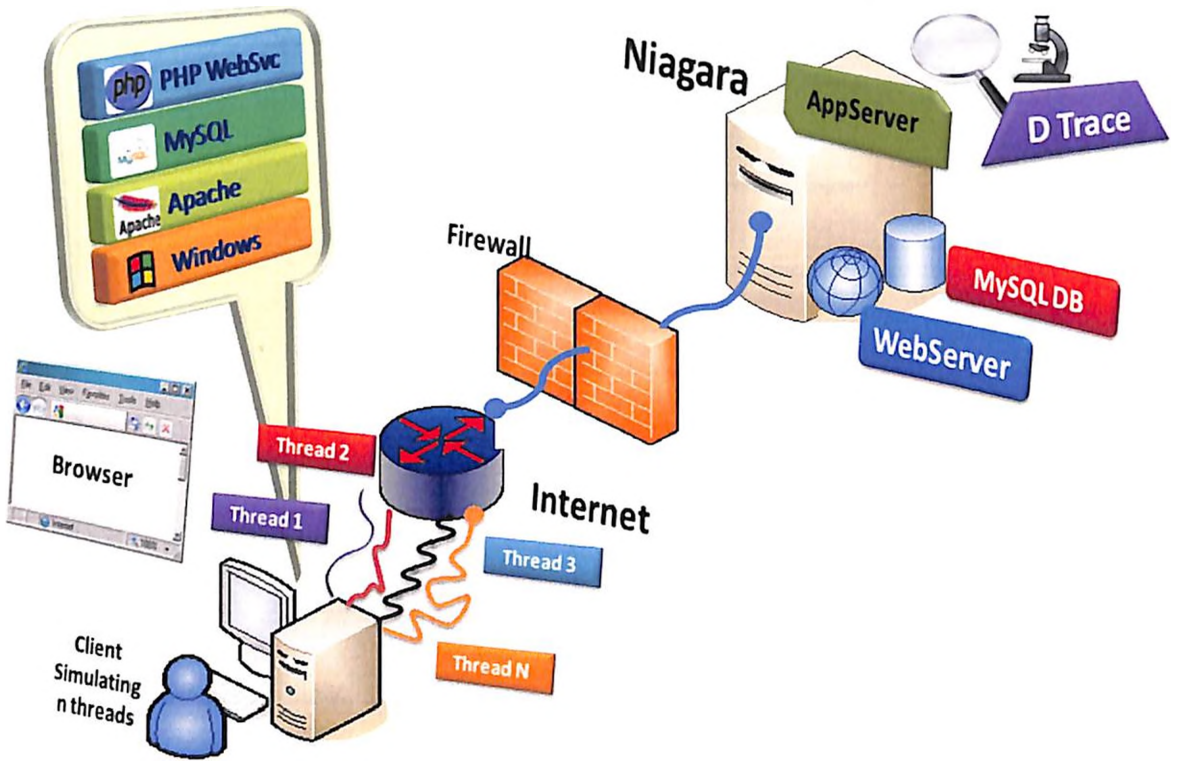


Figure 5-26 Web service Setup without COAF

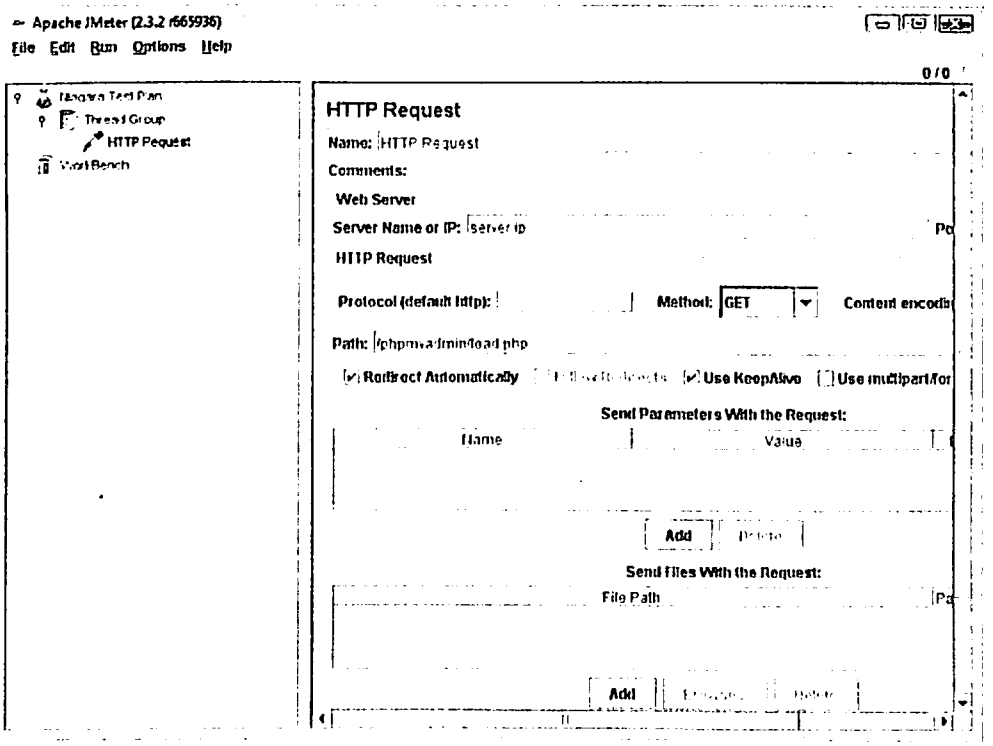


Figure 5-27 Apache JMeter settings for loading queries on the enterprise application

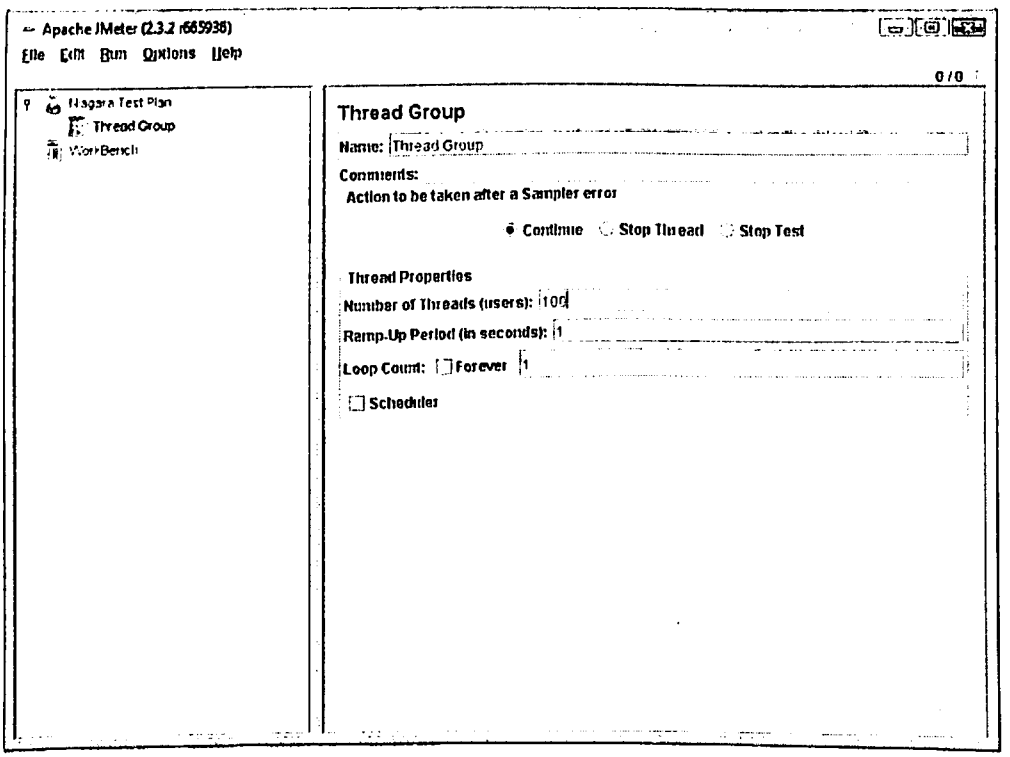


Figure 5-28 Setting up threads and Ramp-up in JMeter for Web Services

Table 5-21 Results and Characteristics of WS NoCOAF

No of Cores	QueryID / Threads	Clusters	No of Command executions		No of Cache Misses		CCE	IPE
			SQL	NonSQL	SQL	NonSQL		
08	1000	42	2,879	1,157	17,301	82,316	71.3	78.3
08	2000	42	3,272	2,271	19,101	149,814	59.0	68.2
09	1000	47	3,043	1,015	17,770	65,922	75.0	81.0
09	2000	49	1,947	1,534	12,357	118,987	55.9	73.3
16	1000	51	4,116	1,351	26,348	64,608	75.3	79.3
16	2000	57	5,604	2,222	44,972	129,573	71.6	66.4
17	1000	59	4,426	1,413	28,908	84,630	75.8	75.6
17	2000	41	7,094	2,980	86,997	128,745	70.4	60.5
32	1000	54	4,611	2,298	50,200	79,131	66.7	71.9
32	2000	67	3,943	2,566	125,999	114,416	60.6	57.2

Table 5-22 Correlation of number of cores against number of clusters

Deployment configuration	Application test case	Pearson's Correlation coefficient
Client-Server	Query1	0.967
Client-Server	Query2	0.922
Client-Server	Query3	0.666
WS No COAF	Queries 1, 2, 3	0.707

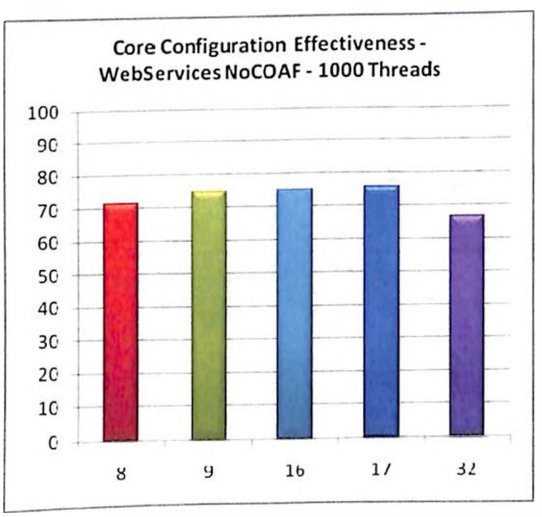


Figure 5-29 CCE for 1000 threads across cores

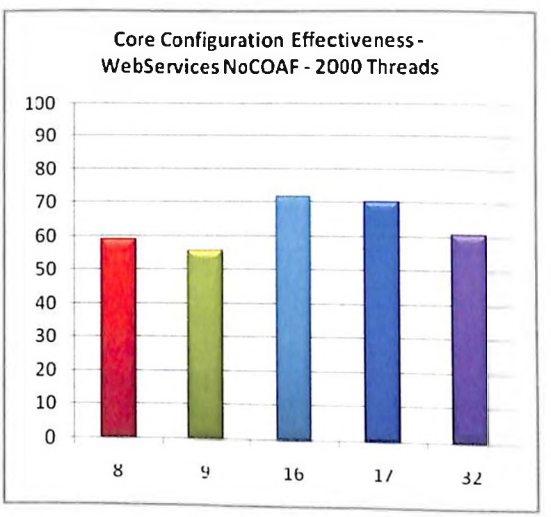


Figure 5-30 CCE for 2000 threads across cores

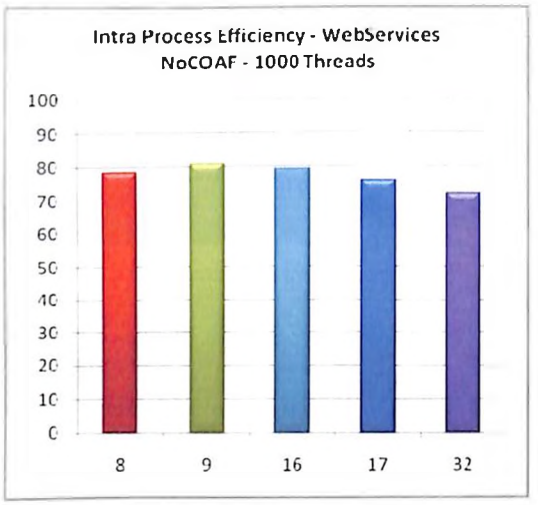


Figure 5-31 IPE for 1000 threads across cores

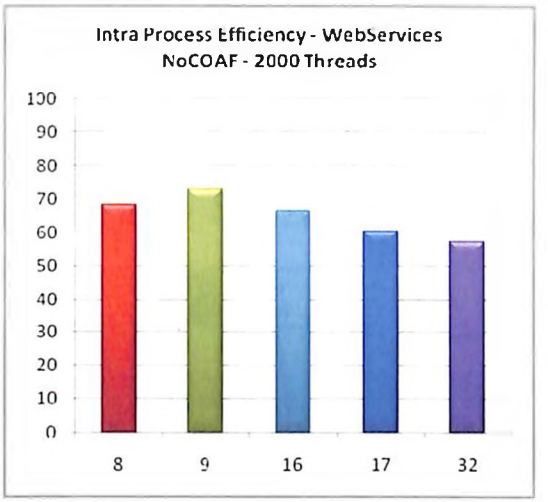


Figure 5-32 IPE for 2000 threads across cores

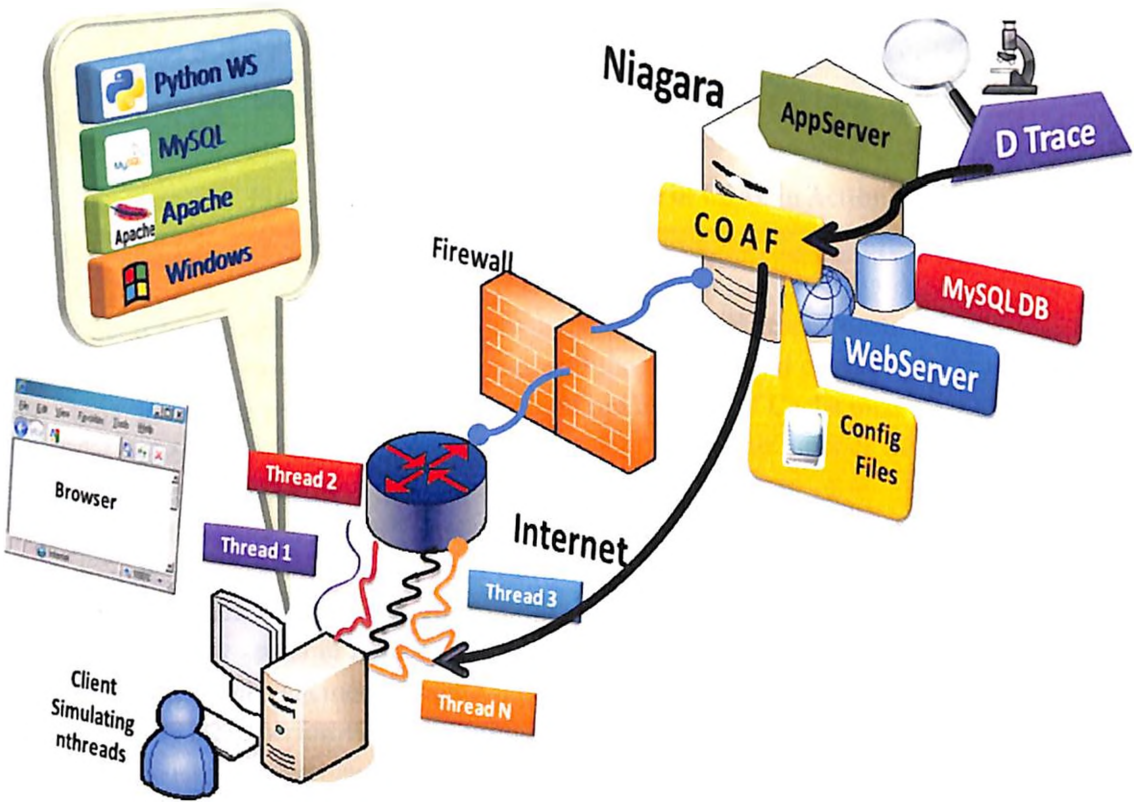


Figure 5-33 Web service setup under COAF

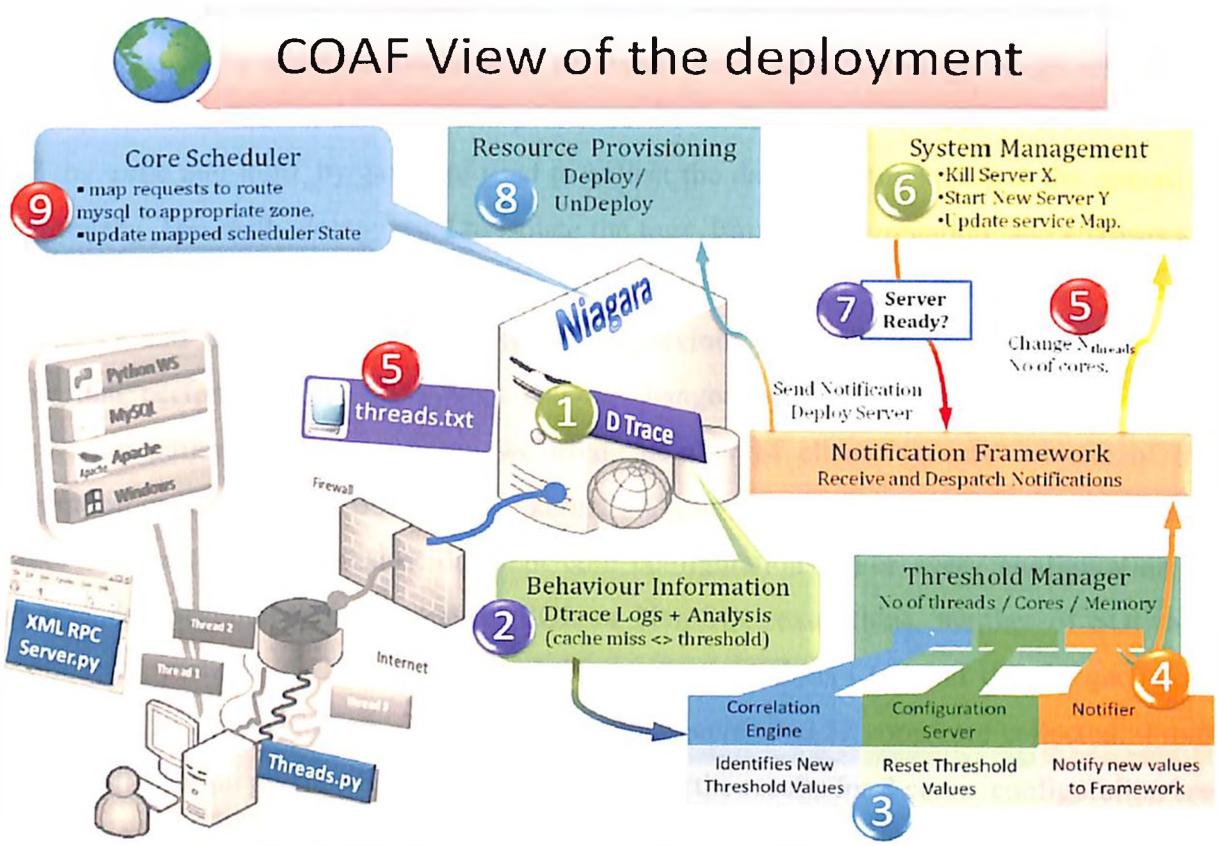


Figure 5-34 Stepwise Perspective of COAF in Action

S No	COAF: Server Perspective for the first iteration
1	DTrace Probes begin logging the performance of the cores in the respective log files. The Cache misses and the system calls are noted across in the respective logs.
2	The raw data from the logs is filtered, and the analysis and clustering is performed on these logs, giving the performance of the current run in terms of Cache misses.
3	This Cache miss with system calls, is used to refine the number of threads as well as the threshold values in the configuration files (in /etc).
4	The performance depends on the run time parameters i.e. number of client threads stored in threads.txt. These newly computed values are notified as **feedback to the system.**
5	The Notification Framework module now relays to the System Management module – either to start a new server or reduce number of threads based on the Cache misses.
6	The System management module, takes the notification (deploy service, start server, stop, undeploy etc.), performs the actions and inform the Notification Framework module.
7	The System Management modules also despatches the status of the server to the Notification Framework module periodically.
8	The Resource Management module ready to take the processing requests.
9	The Core Scheduler schedules the processing requests to the newly provisioned server. Maintains SchedulerState and ServiceMap.

Cache miss is the observed kernel level parameter from the DTrace logs. Logs relevant to cache misses are collected, analysed and presented for feedback. Two probes of DTrace – *viz.* syscall_by_proc and minf_by_proc are used to collect the data from the kernel at the operating system level. The methodology used to isolate the logs, filtration, aggregation, and analysis are same as the methodology followed in Phase II; this methodology is presented in Appendix II, Appendix IV and Appendix V respectively. From previous phase experiments we were able to observe that performance of the system can be changed by changing the application level configuration parameters; for example we used "number of client requests" as one of the configuration parameter. We observed the kernel's Cache miss behaviour for various values of "number of client threads" against different core configurations. For every configuration of client threads, the number of clusters of SQL or Non-SQL executions, number of SQL and NonSQL calls, number of Cache misses and the two computations CCE and IPE for each core configuration (16 cores, 17 cores, 26 cores, 28 cores, 30 cores and 32 cores) are collected. From the core configuration perspective, it could be noted that the results for 8 cores configuration are not presented. In Niagara chip 8 cores are arranged in the same slot. Hence, the results of 8 cores configuration may be an outlier when compared to other core configurations. Since the focus is to establish the awareness of interconnect and its impact on performance, the analysis and results are presented for 16, 17, 26, 27, 30 and 32 cores configurations. The baseline results collected from previous tests that are web services tests without COAF, are taken as the initial baseline threshold values. Core Scheduler module has the handler called COAFConfigurator, which is used to setup the number of client side threads and dispatch those requests to the system. The initial value for the number of threads is taken from web services without COAF baselines values. The value of the "number of client threads" is stored in the threshold configuration file called "threads.txt", which the COAFConfigurator script handler refers to for configuring the number of threads.

From Phase III experiments, we also observed that the number of cache misses were low when the application configuration values for "number of client threads" were around 100; even if the values for "number of client threads" were set at 1000, 2000 etc. the number of client web service requests that got processed by the application was around 110. This provided the first base line information about the range of values for "number of client threads". This helped us to

focus on understanding the behaviour of the kernel for various values of "number of client threads" around 100. This in turn would help us to setup threshold values for the system for this configuration parameter "number of client threads". The initial value is set to "50 threads" and the seed value of 5 threads added in both directions like 50, 55, 60 etc. in the positive direction and 50, 45, 40, 35 etc. in the negative direction. The scripts are configurable to modify the seed value from 5 threads to any number of threads; similarly, the initial value of threads can be modified from 50 to any other value. This setup is done, to understand the behaviour of the system for varying the number of threads configuration observed through the number of "Cache misses". The Cache miss logs are analysed and the results demonstrate the impact of changes in configuration and thus demonstrate an ability to inject the knowledge of Infrastructure to the COAF based system. The iteration of analysing Cache miss logs is continually done for various core configurations to identify a pattern / trend to arrive at the threshold for minimum number of Cache misses.

5.7.2 Phase IV – Observations on multiple Core Configurations

Table 5-23 captures some of the interesting observations obtained for Web services setup with COAF for various number of client threads from 35 threads to 110 threads. This table does not show all the results for all the values of "number of client threads", but it captures important aspects of the system behaviour. The following are key observations one could make from this table:

- Kernel behaviour can be easily represented through CCE and IPE when compared to digging massively large textual logs.
- For a same Core Configuration, the kernel behaviour is different for different values of "number of client threads" for the same application. In other words, application configuration can be changed to get the best results from the underlying system, without modifying the system configuration (source recompiling, rewrite etc.)
- For the same value of "number of client threads", the kernel behaviour is different for different Core Configuration. In other words, system configuration can be changed to get the best results without modifying the application configuration.
- Best configurations for the application and core configuration can be combined to get the best results.

Table 5-23 Results and Characteristics of WS COAF

Number of Cores	Number of Client Threads	Number of observed Clusters	Number of Command executions		Number of Cache Misses		CCE	IPE
			SQL	NoSQL	SQL	NoSQL		
16	35	26	9,237	5,775	14,862	29,241	61.5	89.8
16	50	32	3,639	908	15,262	21,247	80.0	90.6
16	60	34	5,738	1,418	18,173	11,043	80.2	92.0
16	100	41	7,602	1,469	27,811	35,143	83.8	85.0
17	35	33	6,343	5,269	21,113	16,990	54.6	89.3
17	50	32	4,765	3,480	20,104	22,912	57.8	88.5
17	60	32	5,569	4,381	21,393	17,961	56.0	89.1
17	100	10	8,944	9,927	9,506	18,540	47.4	93.0
26	100	34	4,358	3,426	31,310	16,300	56.0	86.8
26	110	33	3,902	1,072	29,649	22,880	78.4	86.3
28	100	36	6,369	4,621	34,842	27,450	58.0	83.8
28	110	39	7,248	6,472	39,188	31,757	52.8	81.7
30	100	32	7,181	1,981	32,908	33,953	78.4	83.7
30	110	41	4,201	2,702	39,456	47,045	60.9	79.1
32	45	6	7,147	9,472	8,943	18,964	43.0	92.8
32	50	2	10,895	8,201	25,383	404,535	57.1	50.5
32	100	3	25,701	26,172	298,719	314,077	49.5	34.3
32	97	6	6,030	5,520	9,784	14,940	52.2	93.4
32	105	6	6,240	5,261	10,158	10,130	54.3	94.2

5.7.3 Phase IV – Performance can be changed by modifying application configuration

This section highlights the importance of COAF based feedback. In a normal enterprise application administration, the configuration settings are usually setup for either default values or maximum utilization. For example in the case of number of client requests that can be processed from the system, if choice is between 1000 or 2000 for number of client requests to process, then the obvious choice for the administrator is to set them for high values at 2000. However, we have observed that the system cache-misses are high for those settings, and we observed low cache misses happen at settings around 100. Figure 5-35 and Figure 5-36 illustrate the differences between the performances with COAF based feedback versus performance without feedback (NoCOAF) for a 16 cores configuration system. Figure 5-35 compares CCE values (that is number of schedules of MySQL commands) for 16 cores configuration with and without COAF feedback. For the variable "number of client threads" the performance is better with COAF for the values 50, 60 and 100 than NoCOAF values (1000 and 2000), except for a setting of "35 threads". This indicates the possibility of achieving better performance by just varying the values for "number of client threads" for a given core configuration. As we can observe from Figure 5-36 for setting of 50, 55, and 60 client threads the IPE value is higher. This implies that the number of Cache misses is relative low (or high cache hits in other words) within each schedule (time slot) given by the operating system. For 16 cores configuration, 60 threads would be a good setting based on the above results obtained from the logs for optimum cache hits / lower Cache miss.

Without COAF feedback (i.e. NoCOAF values from the figure), default values would have been set for 1000 or 2000 – as we are interested to process maximum number of client requests. However, we have observed that values around 50, 60 or 100 are good. This observation is important for emerging new subscription and billing models of enterprise cloud hosting scenarios. Assuming the subscription is based on the number of client requests that can be processed and if the provider bills the enterprise based on the "number of client threads" (because that is the only human configurable value), then the cost of that service would be arrived for the settings 1000 or 2000; We have observed that even though the system has subscribed to 1000 or 2000 client requests, the system can only process upto 110 requests efficiently for a given 16 cores configuration. With this COAF based observation, we can say the

setting could be 110 and not 1000 or 2000; and the charges appropriately reduced to $1/10^{th}$ or $1/20^{th}$ of the original planned cost respectively.

5.7.4 Phase IV – Performance can be changed by modifying Core configuration

This section highlights the observations due to next dimension of variability related to Core configuration. Results are presented in Figure 5-35 to Figure 5-40 for three different core configurations namely 16 cores, 17 cores and 32 cores; the results compare the CCE and IPE values for six different thread settings, 1000, 2000 (the default settings without the COAF feedback) and 35, 50, 60, 100 (the different settings tried with COAF feedback This is to compare the behaviour between

- 16 cores to 32 cores (what happens if the compute capacity is doubled) and
- 16 cores to 17 cores (for giving extra core with interconnect)

32 cores configuration has completed the application jobs (MySQL) in 33% lesser schedules when compared to 16 cores configuration; in other words, 32 cores configuration has completed the same job in 33% faster than 16 cores configuration; this directly demonstrates the linear scalability of increasing the core. However, the scalability is not double (ideally 100% increase) for doubling the cores from 16 to 32. Instead it has given only 33% more efficiency.

17 cores configuration has just one more core when compared to 16 cores configuration. Additionally 17 cores configuration has core-interconnect overhead when the data overflows to caches that across core-interconnect. When compare the results using IPE, we can see 17 cores configuration has actually performed very well when compared to 16 cores configuration, despite the overhead due to interconnect. This observation corroborates with the Phase II observation from syscall_by_proc and dispqlen_by_cpu probes, which indicates that operating system calls are scheduled in 17^{th} core, where as the application calls (MySQL and Non-MySQL) are scheduled in remaining 16 cores. Without the COAF feedback, administrator would choose 16 cores configuration as it is symmetric and without any interconnect overhead. However, with the COAF feedback, administrator would choose a 17 cores configuration.

32 cores configuration has performed extremely well when the values are 45, 97 and 105 for the variable "number of client threads". This indicates the threshold could be between 45 to 105, and can be setup in this range, and the threshold can be further fine tuned for well-defined workloads.

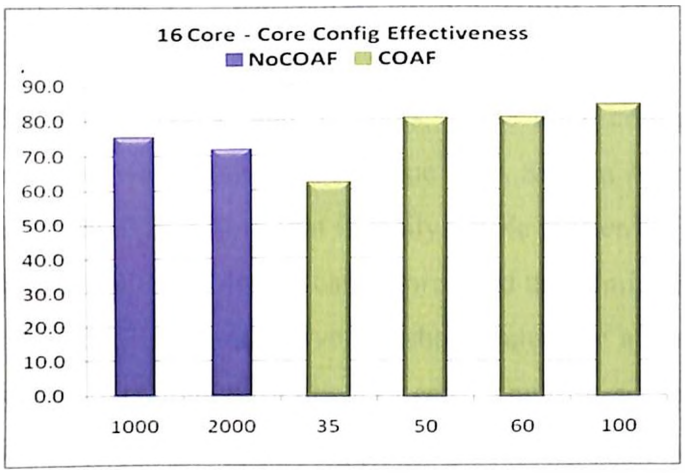


Figure 5-35 16 cores CCE (No COAF vs COAF)

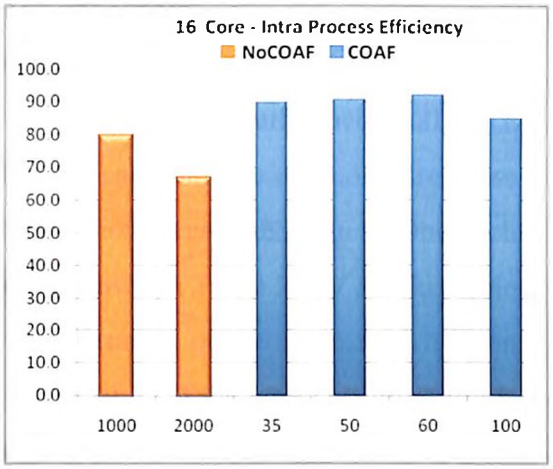


Figure 5-36 16 cores IPE (No COAF vs COAF)

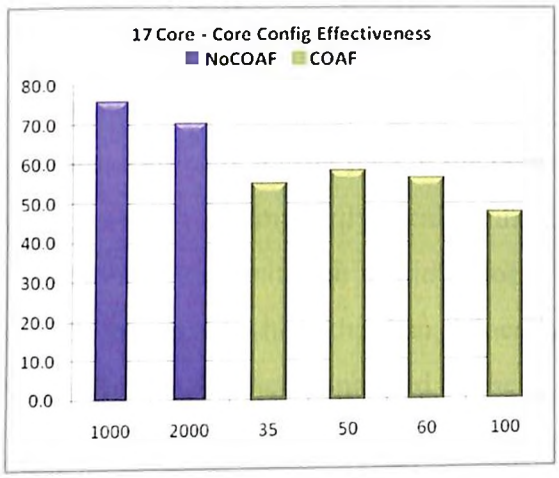


Figure 5-37 17 cores CCE (No COAF vs COAF)

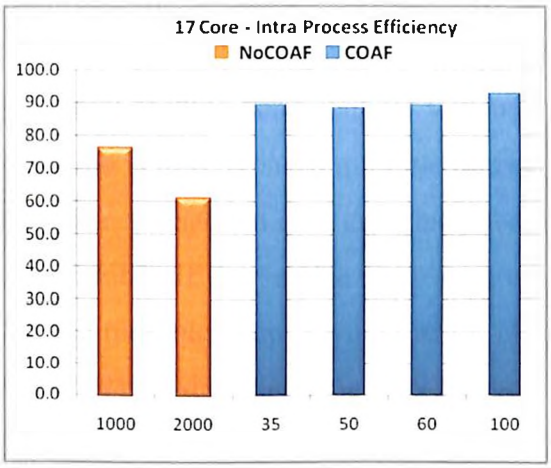


Figure 5-38 17 cores IPE (No COAF vs COAF)

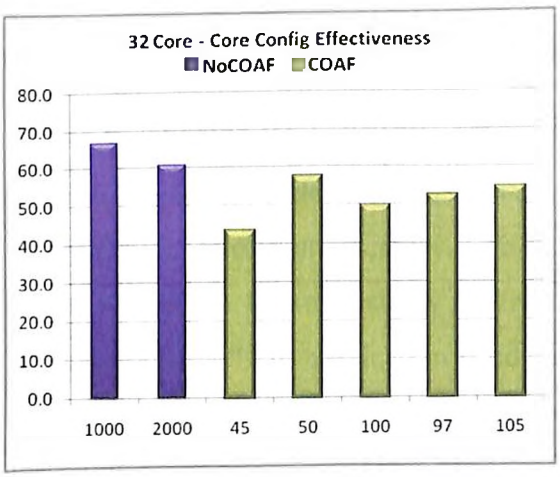


Figure 5-39 32 cores CCE (No COAF vs COAF)

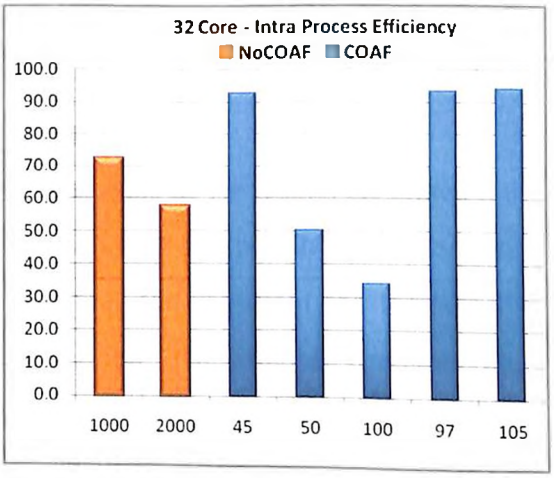


Figure 5-40 32 cores IPE (No COAF vs COAF)

5.7.5 Phase IV – Enterprise Administrator's view of configuration

Figure 5-41 and Figure 5-42 provide another view of the results comparing the performance of 26 cores, 28 cores and 30 cores. These three configurations in general performed well for a value 100 when compared to value 110. System Administrator can combine and compare these observations in a form that is easily usable for her/his configuration thresholds values. One such sample look-up table that can be prepared the administrator is shown in Table 5-23. As this table shows, this is an indicative threshold value for a system that runs a query against 10 million records, the "number of client threads" could be set for a value of 35 if the core configuration is 16 cores, and for the same application, the value of "number of client threads" could be set for a value of 100, if the core configuration is 28 cores .

5.8 PHASE V - COAF TEST BED SET UP – TEMPORAL COHERENCE STUDY

The objective of the Phase V experiment is to study temporal caching behaviour for various configurations. Typically master data in the enterprise application is re-referenced multiple times and thus represents the temporally local nature of the data. The transaction data especially for CRUD operations represents the spatially local nature of the data. If the service requests are dispatched to the server where the data is locally available and reusable, then it would result in improved performance when compared to the dispatch to the server where the data may not be locally available. This aspect of contextually dispatching the request is tested for various core configurations in this phase. The results of this experiment become very interesting considering the emergence of new pluggable PCI bus based flash memories [231] that can fit larger memory sizes in the range of tera bytes of data near to main memory.

5.8.1 Phase V – Temporal Application Parameters – Setup

The Phase V test case is designed in such a way that the data size of the results generated out of the query can fit into main memory. These kinds of test cases are possible in enterprise environment, where the lookup data is relatively static and that can represent the temporally predictive scenarios. MySQL application is hosted as a web service. The database contained the dataset related to various account numbers; these account numbers are generated with orderly sequenced numbers and with randomly generated numbers. The distribution of account numbers with a mix of orderly created and randomly generated values thus, removes the homogeneous nature of data, and resembles enterprise transaction data. The application query uses account

number as the primary search criterion. Enterprise applications are predominantly read only in nature: therefore, if memory size could fit the standard master data combined with transaction data, then the data would represent better spatio-temporal characteristics when compared to the data size that can overflow the memory size. The "number of client requests" is used as the controlling variable and the experiment is done to test the execution behaviour of the application for various values for this variable. 35, 50 and 60 are the values set for "number of client requests" and results are captured for 16 cores, and 17 cores configuration. Similarly, the "number of client requests" is set with values of 100 and 110 threads and the results are captured for 26, 28 and 30 cores configurations.

5.8.2 Phase V – Cache memory size could influence the performance

Table 5-25, represents the consolidated results for 8 core, 16 cores, 17 cores, 26 cores, 28 cores and 30 cores respectively. From the results, it can be observed that all the application processes run within 4 clusters for 16, and 17 cores configurations with reduced Cache misses. This corroborates with our earlier experiments of previous phases that Operating system is smart enough and fair in scheduling the well-prepared processes (spatially and temporally coherent processes) for execution.

For interconnect configurations such as 26, 28 and 30 cores configurations, the number of clusters has increased; this indicates the occurrence cache misses and/or context switches. Ideally, size of data may not be a reason for cache misses / context switch as the data is in such size as fits into memory; thereby data is temporally and spatially coherent. However, operating system sees almost large numbers of cores across interconnect. When the configuration is 17 cores, Operating system is intelligent to identify 16 cores on one side of the interconnect, and one core on the other side of the interconnect. However, when Operating system sees 26, 28, and 30 core configuration, it sees that the number of cores are spread across the interconnect; because, the Operating system is fair in its scheduling, it chooses to distribute the load across cores. While the execution is distributed, the data is present in the cache that is on the other side of interconnect. Since, the cache is shared Operating system first tries to fetch the data across interconnect. This aspect of fair scheduling policy of Operating system across multiple cores, modifies the spatially local coherency of the data, which resulted in Cache misses. Hence, the

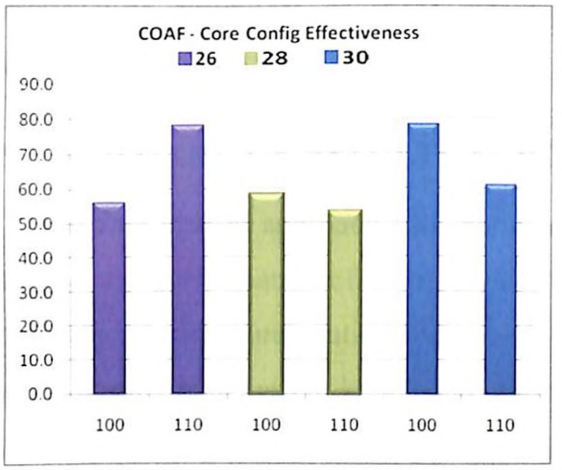


Figure 5-41 CCE for various core configurations

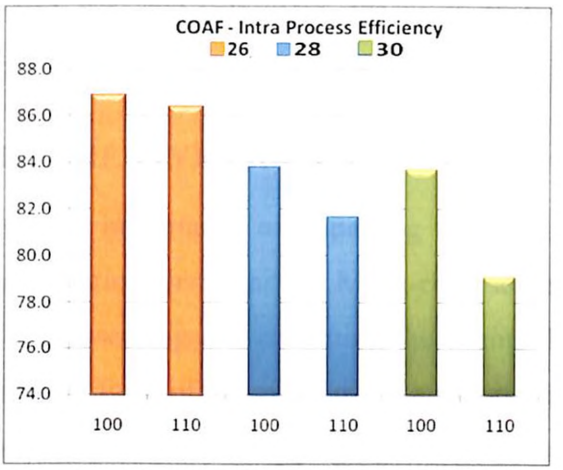


Figure 5-42 IPE for various configurations

Table 5-24 Typical Configuration recommendation for Administrator

No of Cores	Approximate number of records to process	Recommended number of Client Requests
16	10,000,000	35
17	10,000,000	35
32	10,000,000	45
26	10,000,000	100
28	10,000,000	100
30	10,000,000	100

Table 5-25 Results of temporal Characteristics for in-memory

No of Cores	Number of Client threads	SQL Runs	SQL cache misses	Non SQL runs	Non SQL Cache misses
08	50	4	26	38	594
16	35	9	92	81	3,577
16	50	4	7	96	3,632
16	60	4	63	151	5,675
17	35	4	74	172	6,269
17	50	4	54	133	4,711
17	60	4	61	164	5,508
26	100	16	76	96	4,282
26	110	4	21	77	3,881
28	100	27	1,748	151	4,621
28	110	27	1,997	167	5,251
30	100	27	1,911	152	5,270
30	110	8	46	90	4,155

Infrastructure awareness of cache memory size and the cache location are important for improved scheduling.

5.9 SUMMARY OF RESULTS ACROSS ALL THE FIVE PHASES

The five phased approach to experiments shows the importance of imparting feedback to improve the performance of web services based application that runs on Multi-core servers. Comprehensive instrumentation framework is used for observing and collecting the events that happen at operating system kernel and storing them in the form of logs for further analysis. We have developed an extensive workflow methodology (extract, filter and aggregate relevant events such as cache-misses, the number of schedules of a specific process etc.) to extract relevant events from DTrace logs. These events are abstracted by two metrics namely Core Configuration Efficiency (CCE) and Intra Process Efficiency (IPE). CCE represents effectiveness of scheduling the desired application processes on the core when compared to other processes. IPE represents the efficiency of the execution of these processes. Cache-miss is used as the underlying parameter that is used to arrive at the values for CCE and IPE. The concepts of Connection-Orientation and Infrastructure Awareness are demonstrated through multiple experiments from Phase II through Phase V. Following are the key points that have been demonstrated to achieve the objective of COAF.

1) There are tools and methodologies that can be leveraged to monitor and measure the kernel level parameters in non-intrusive manner.
2) It is possible to deploy the Multi-core servers with various Core configurations such as 2 cores, 4 cores, 16 cores, 17 cores, 32 cores etc.
3) Cache-misses directly correlate with performance of the system and it can happen due to spatial and temporal incoherency.
4) The data gathered at the kernel level is used to relate and associate the deployment configuration parameters of the system at run time.
5) Application deployment models affect the spatio-temporal behaviour of the application.
6) Operating system is fair in its scheduling.
7) Operating system schedules the processes according to number of cores available for processing.

8) Process queues are evenly spread across cores when multiple cores are available for processing
9) Loading pattern of the processes varies from one core to another core within the multi-core configuration..
10) Operating system gives priority to well-prepared processes (which has both data and execution instructions) compared to ill-prepared processes.
11) It is possible to predict the behaviour of the operating system, using application deployment configuration parameters
12) For a given Core Configuration, the kernel behaviour is different for different values of "number of client threads" for the same application. In other words, application configuration can be changed to get the best results from the underlying system, without modifying the system configuration (source recompiling, rewrite etc.).
13) For the same value of "number of client threads", the kernel behaviour is different for different Core Configuration. In other words, system configuration can be changed to get the best results without modifying the application configuration.
14) Best configurations for the application and for the core configuration can be combined to get the best results.
15) By making the application connection aware, it is possible to modify the number of process schedules at the operating system level, without modifying any operating system parameters. This means that desired processes get more attention from the operating system, during scheduling.
16) It is also possible to increase the efficiency of execution, if the knowledge of infrastructure is used to set the run time parameters for the enterprise infrastructure. This means that a desired process can work in a very efficient manner during execution.
17) The importance of an active feedback mechanism using two derived parameters CCE and IPE are substantiated. Active feedback is used to get better run-time performance of the applications on Multi-core systems.
18) The enterprise administrator can now have a composite chart of empirical correlations that identifies and set thresholds values for various applications. The threshold parameters can have preset or new threshold values. Alternately, all hardware and

software configurations in the enterprise can be modelled as web service resources and the parameters for deployment set accordingly.

19) Notifications can be processed independent of the application as the instrumentation processes are non invasive.

20) Other Infrastructural component like Cache memory size could influence the spatio-temporal characteristics of the application.

21) The above observations help the enterprise administrator, to setup and configure the application as well as the hardware configurations in the enterprise - effectively and efficiently.

22) The test for the Core Configurations on the Web service based platform with and without COAF lead to statistically significant results as detailed in Appendix IX – rejecting the null hypothesis that it is not possible to control the performance at the application level from kernel level observations.

6 CONCLUSIONS

This chapter concludes the thesis with the summary of lessons learnt and experiences derived from the implementation. This research addresses the critical issue of using Multi-core for enterprise class web service based applications. It elicits the hardware and software initiatives for Multi-core along with their shortcomings.

It further extends the advantages and proposes the marriage of these hardware innovations of Multi-core to the web services based software architectures. This proposal is the new architecture called COAF. COAF focuses on Enterprise Computing using Multi-core servers for web service based applications. COAF can efficiently utilize the abundant computing power provided by Multi-core processors to compute the workloads of web service based enterprise applications.

6.1 CONTRIBUTIONS OF THE THESIS

COAF is a first step in the direction of the adoption of Multi-core servers in the enterprise. It is an architectural framework for designing and deploying web services based enterprise applications on Multi-core platforms without modifying the applications. The distinct ability of COAF is to pass the application context that is available at both design time and run time to the operating system through deployment configuration methods, so that operating system is correctly positioned to leverage the underlying hardware threads.

COAF proposes a six module approach. This six module approach standardizes the architectural methodology to deploy web services based enterprise applications on the Multi-core servers, similar to how a three layer architectural methodology standardized the development and deployment of web based applications.

The six modules are Resource Provisioning module, Behaviour Information module, Threshold Configuration module, Core Scheduler module, System Management module and Notification Framework module. Each module has a well-defined responsibility and the design. This division of six key responsibilities also standardizes the deployment of Multi-core in the enterprise.

Two architectural concepts namely, "Connection-Orientation and Infrastructure Awareness" are identified to derive the best of admission control and feedback mechanisms. "Infrastructure

Awareness" enables dynamic_models of service behaviour with a feedback mechanism that can enable the dynamic changes that are required in resource allotments under varied loads. "Connection-Orientation" is obtained by contextually grouping similar services, thus providing a scope for data alignment and reduced Cache misses. Two important data structures namely ServiceMap and SchedulerState are used to group and contextually dispatch the service request to the appropriate core configuration and thus assure the Connection-Orientation to the system.

"Cache miss count" is identified as the key metric for measuring the performance at the operating system level. Two system parameters are derived from "Cache miss count" namely, "Core Configuration Effectiveness" and "Intra Process Efficiency" to quantitatively describe the performance at application level for a given core configuration.

Figure 6-1 presents the overview of a COAF Enterprise from an enterprise administrator's perspective.

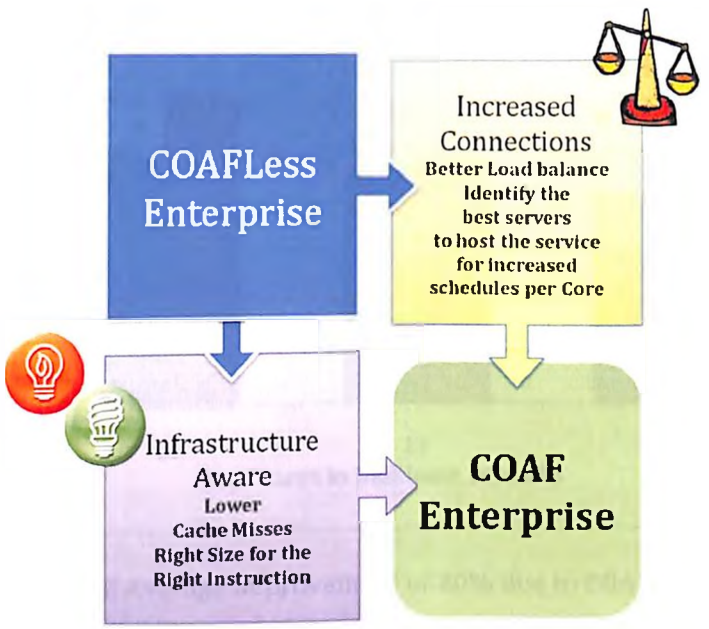


Figure 6-1 From a COAFless Enterprise to COAF

Enterprise data is empirical by nature. COAF extracts the run time knowledge of the application by observation at the kernel level. This run time knowledge is correlated to application deployment configuration parameters, thus making application tuneable. Thus COAF marries the actual performance model of the hardware at run time to that of the configurable parameters of software, thereby encouraging an active feedback between the hardware and software infrastructures.

With an enterprise test bed set up, a typical prototype of COAF is demonstrated for a standard enterprise class application. The six modules of COAF are built into this prototype on a Niagara based Multi-core server. The hardware platform utilises the native DTrace non-invasive probes to collect run time information as in the production environment. The power of COAF is highlighted through the usage of the two derived parameters *viz.* Core Configuration Effectiveness and Intra Process Efficiency. Control mechanisms are also provided in the prototype. The experiment is conducted in five phases to elicit the advantage of deploying COAF methodology stepwise. The prototype demonstrates variation in performance for different deployment configurations, by varying application level parameters such as number of application threads. The average performance improvement across all the experiments is 20% by improving the small percentage of desired process execution as shown in Figure 6-2.

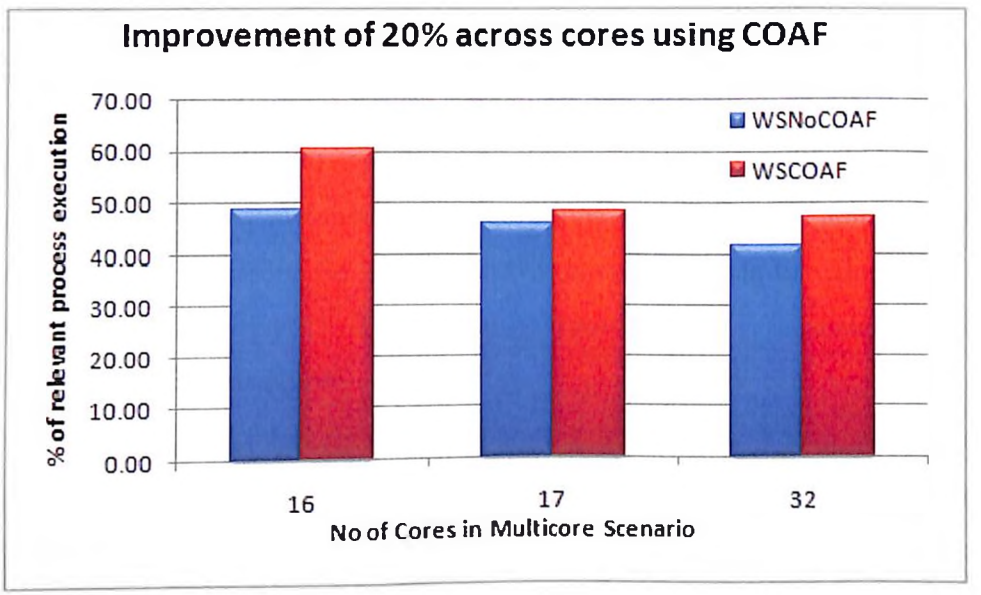


Figure 6-2 Average improvement of 20% due to COAF

The implementation demonstrated that the efficiency of the overall system can be improved upto 38% (for 16 cores configuration setup), by just modifying deployment parameters (viz. the number of threads to 60 from the default setting of 2000 threads). The prototype also demonstrated the optimum utilization of cores (even though 32 cores were available in the setup, the application used only six cores for scheduling the process and the remaining 26 cores were not used) thereby releasing the unused licenses of software. This information is extremely meaningful when the software is licensed and the return on investment computation is based on number of cores used. The implementation also showcased improved results for temporal

alignment as well as symmetric vs asymmetric configurations. This average improvement of 20% to 38% in the system performance due to improvement in relevant schedules and decreased Cache misses corroborates with the well known theoretical square root limit determined by Chow [50] and Hartstein [110] (41.42%), which is, if the cache size is doubled, the miss rate drops by the factor of square root of two.

6.1.1 Benefits of COAF

Enterprises follow strictly governed compliance processes. To accelerate the adoption of Multi-core in the enterprises, COAF embraces non-disruptive adoption methodologies. COAF encourages a continuously improving tuning model. This is extremely significant from the enterprise perspective as this approach leverages the existing investments in single core infrastructures and new innovative Multi-core infrastructures.

Modelling the software and hardware as Web services based Resources, enables the pluggability of COAF. This pluggable nature empowers the enterprise architect to embrace different implementations infrastructure such as hardware, operating system configurations, application server configurations and database configurations etc. Enterprise administrators can directly use the large data available for setting and modifying threshold values for various deployment configuration parameters. This dynamic provisioning is especially useful for effective hot deployment of Web services.

COAF uses fair scheduling methods of the operating system and hence concentrates on refining the inputs provided to the operating systems from application context. It is important to note that COAF does not supplant any caching or scheduling algorithms at the operating system level, but rather complements web applications by offering useful parameters to increase the performance of the application.

Conceptually, six layer framework of COAF is easy to maintain as the role and responsibilities for each layer is well defined and there are straightforward implementations to manage the SchedulerState and ServiceMap data structures. Components of all the layers are pre-existing as off-the-shelf available components and can be leveraged directly in the enterprise infrastructure. The suggested components such as Apache, NaradaBrokering, etc. for implementing the six layers are available open source and are available as free source from the community. Thus, there is no additional cost to the enterprise for implementing COAF.

COAF components such as NaradaBrokering, Apache 2 etc. are inherently scalable can be scaled horizontally with multiple instances deployed for each layer if needed. As all the resources in the COAF are modelled as Web services Resource framework, multiple instances of resources can be plugged into the system depending on the usage and scalability needs.

Enterprises have huge investments for system administration. Enterprise administrators manage and monitor the systems and tune those systems using various configuration parameters for improved performance.

For example, a MySQL administrator uses "number of threads" as one of the configuration parameter to improve the performance of the database application. This configuration is based on the values generated while calibrating the MySQL application in ideal conditions. When MySQL application is executing the workload, and if the performance of the application is not as desired, then the administrator needs to tune the number of threads at runtime. In this situation, COAF based metrics enable the administrator to quantitatively judge, and identify the appropriate value for the configuration parameter for optimum performance. Once the parameter is setup, the active feedback mechanism of COAF enables the enterprise administrator to verify whether the configuration settings have affected the behaviour of the application to the desired performance. Administrators use this feedback information from the kernel observation and can compute their own index based on the IPE and CCE values. Table 6-1 contains one such representation of the administrators indices based on IPE and CCE for MySQL application. With COAF, the enterprise administrator effectively use the rich knowledge of empirical data to create "design time" threshold values as initial deployment values before deploying the application. Administrator sets up an initial value for configuration parameter "number of threads" as 50 for 16 core configuration setup. During execution of this application, administrator perceives considerable slowdown in the execution. Administrator's choice would be to reduce the workload on the server to improve the performance. Without the COAF based information, and by natural intuition, it is logical to reduce the number of threads to reduce the workload on the server, thereby to increase the efficiency of application processing. In that context the administrator would have modified the settings by decreasing the number of MySQL threads to 35. However, with COAF based information such as in Table 6-1, the ideal setup for 16 core configuration would be 60 threads.

As we can see from the table,

- CCE (the number of relevant application runs) is highest (80.20) when the number of threads is setup for 60 for 16 core configuration, and
- IPE (the efficiency of relevant application run) is highest (that is 92.00) when the number of threads is setup for 60 for 16 core configuration.
- The combined Administrator index is at 65.46.

With the Administrator's Index as well as CCE and IPE in Table 6-1, the administrator understands that it is better to increase the number of threads to 60 rather than decrease the number of threads to 35. As the COAF based metrics are empirical, those metrics can be enhanced and can guide the administrator to setup and refine the threshold value, and which in turn empowers the administrator on his Multi-core deployments.

Among the various core configurations, it is important to identify the core configuration that gives the best performance. Enterprises buy and operate the software on a "per-core license basis". Hence it is beneficial for enterprises to understand the cost or utilization per core. Following is an example that represents the computation methodology for per core utilization. From our experimental results, Table 6-2 and Table 6-3 show the extracted observations relevant to 4 cores and 32 cores configurations; the results for all three queries are summed to generalize the effects of all three queries. From Table 6-3, it can be seen that the 4 core configuration has an IPE is 82.9%. That means in the four cores, 82.9% of the instructions are cache hits and 17.1% result Cache miss. For the computation purposes, let assume a cache miss is 100 times costlier than a cache hit. When this aspect is factored into calculating the efficiency of the scheduling, the percentage of time spent on "useful scheduling to application" is only 4.62%. Similarly, for 32 cores configuration, IPE is 32.9% and the Cache misses are 67.1%. The percentage of time spent on "useful scheduling of application" is 0.48%. This is shown in Table 6-4. This means the speedup from 4 cores to 32 cores is about 18.5% more. This result does not reflect linear scalability that would have ideally 8 times more at 800% instead of 18.5%.

If we compute the IPE along with respective speed-ups, the summary would look as illustrated in Figure 6-3, where it is observed that the performance of:

- 4 Core > 5 Core,
- 8 Core > 5 Core and 9 Core,
- 16 Core > 9, 17 and 18 Core, and

- 32 Cores > 17 and 18 Cores.

Thus, CCE and IPE help in calculating the utilization per core.

Operating System schedules the processing request to the Core. The actual number of cores used is less than the numbers of cores were available. Table 6-5 summarizes one such observation obtained for web services based deployment without COAF feedback. In this table, the number of cores and the configuration parameter "number of client threads" are set; the actual number of threads used for execution and the actual number of cores used for execution is also listed. It can be observed that the operating system did not allocate more than 6 cores even when there are more number of cores (9, 16, 17, and 32 cores) available for execution; also, the maximum number of threads used for processing cores did not go beyond 20, even though the application was configured for 2000 client threads usage; both these observations are essential in planning more appropriate configuration. When the cores are not used, then license fees and Annual Maintenance Charges for the software for those unused cores can be saved.

The ability of COAF to understand the system behaviour and associated parameters such as CCE and IPE helps to enhance the performance of newly evolving Multi-core systems such as cloud and grid, and specialized applications like weather modelling, gene search etc. Best practices of COAF such as setting threshold values for each parameter, Connection-Orientation upto the compute end point, and context awareness of subsystems such as cache size, main memory size etc. complement and enhance the performance of these applications. For large scale cloud and grid deployments special code implementations such as Google File system, Amazon Web services, Cassendra etc. are developed to leverage the power of Multi-core capabilities. These implementations have similar goals like COAF based systems, such as optimum performance, thread scalability, memory management etc. While these systems are newly custom developed, COAF based systems are just leveraging existing systems including operating systems and application servers.

In conclusion, we are able to demonstrate that it is possible to:

- Change the performance of web service based enterprise applications from the application layer by contextually dispatching request using the data structures ServiceMap and SchedulerState, and the concepts of Connection-Orientation and Infrastructure Awareness.

- Leverage the performance of Multi-core by changing deployment configurations and demonstrated using various core configurations such as 8, 16, 17, 32 cores etc.
- Alter the performance of an application, by observing the kernel and hardware parameters without any modification to the operating system, and demonstrated this ability using the "Cache miss" metric and the derived parameters CCE and IPE.
- Affect the performance of a generic application in Multi-core system without changing the operating system configurations and demonstrated the change in performance by changing the number of application client threads.
- Change the Spatio-temporal characteristics due to the awareness of infrastructure and demonstrated the concept with improved performance for data sizes that fit into cache memory, when compared to datasets that overflows the cache and interconnects.
- Effectively contain and distribute enterprise applications amongst cores to achieve performance through core symmetricity and core affinity and demonstrated the improved behaviour of an asymmetric configuration (9 core and 17 core) and the potential cost savings by removing the licenses which otherwise accounted for idle cores that are configured but never used by operating system as in the case of 32 core configuration.

Effect changes in the performance of enterprise applications without changing either the enterprise application binaries or enterprise application source and demonstrated an improved performance upto 38 percent and an average improvement of 20 percent by changing the configuration parameter "number of client threads".

Table 6-1 Administrators Indices for MySQL to identify core and thread configuration

	No of Cores	No of Client Threads	CCE	IPE	Admin.'s Index
For a given Service, Admins Choice without COAF	16	35	**61.50**	89.80	60.42
Initial Configuration for the same service	16	50	80.00	90.60	65.18
Administrator enabled with the knowledge of COAF feedback	16	60	**80.20**	**92.00**	65.46
	16	100	83.80	85.00	64.96
	17	35	54.60	89.30	58.21
	17	50	57.80	88.50	59.13
	17	60	56.00	89.10	58.64
	17	100	47.40	93.00	56.03
	32	45	43.00	92.80	54.21
	32	50	57.10	50.50	51.77
	32	100	49.50	34.30	45.01
	32	105	54.30	94.20	58.69
	26	100	56.00	86.80	58.34
	26	110	78.40	86.30	64.09
	28	100	58.00	83.80	58.55
	28	110	52.80	81.70	56.63
	30	100	78.40	83.70	63.63
	30	110	60.90	79.10	58.66

Table 6-2 Summary of 4 core and 32 core data from Table 5-20

No of Cores	QueryID / Threads	No of Command executions		No of Cache Misses	
		SQL	NonSQL	SQL	NonSQL
04	q1	896	954	9,357	30,643
04	q2	151	559	3,942	14,404
04	q3	136	507	1,397	8,515
32	q1	272	1,730	50,020	40,575
32	q2	2,926	1,499	101,732	257,258
32	q3	3,478	426	61,516	150,818

Table 6-3 Summary Extract of Table 5-20 for Phase II 4 and 32 core configurations

Number of Cores	Number of Command executions		Count of Cache Misses		Overall IPE
	SQL	NonSQL	SQL	NonSQL	
04	1,183	2,020	14,696	53,562	82.9
32	6,676	3,655	213,268	448,651	32.9

Table 6-4 Per Core Utilization for 4 and 32 core configurations

Number of Cores	IPE	100-IPE	Useful Time per core	Total useful time = (Number of Cores * Useful time per core)
4	82.9	17.1	4.624	18.4951
32	32.9	67.1	0.488	15.6134

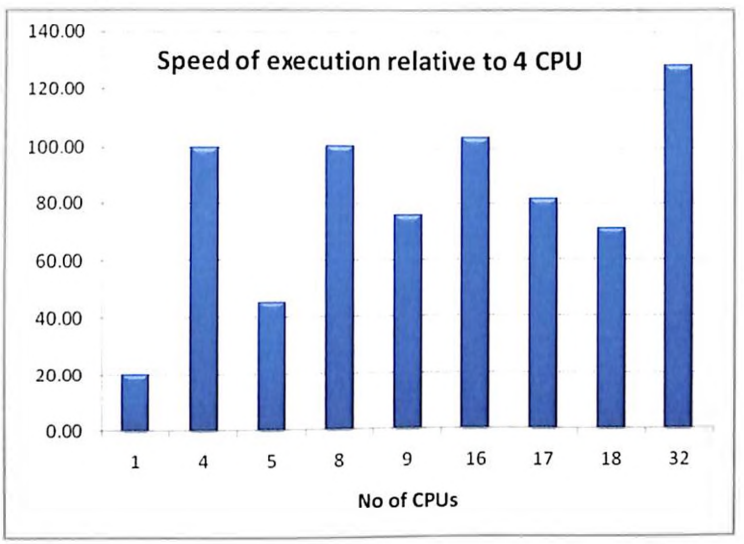


Figure 6-3 Execution speed compared

Table 6-5 Actual cores used against available cores

Type	Available Cores	Number of client threads	Cores used	Core IDs	Max number of Requests processed
WS NoCOAF	08	1000	5	0,1,3,4,5	10
WS NoCOAF	08	2000	5	0,1,3,4,5	20
WS NoCOAF	09	1000	6	0,1,3,4,5,6	10
WS NoCOAF	09	2000	6	0,1,3,4,5,6	20
WS NoCOAF	16	1000	6	1,2,5,6,8,9	10
WS NoCOAF	16	2000	6	1,2,5,6,8,9	20
WS NoCOAF	17	1000	6	5,6,7,10,11,13	10
WS NoCOAF	17	2000	6	5,6,7,10,11,13	20
WS NoCOAF	32	1000	6	11,12,15,16,27,28	10
WS NoCOAF	32	2000	6	11,12,15,16,27,28	20
WSCOAF	32	050	6	4,5,13,22,23,26	5
WSCOAF	32	100	6	4,5,13,22,23,26	10

7 FUTURE WORK

This thesis lays out new possibilities and scope for adopting Multi-core as a hardware platform for a wide range of web service applications. The scope for future work is large, given the range of hardware and software innovations. This chapter focuses on three broad areas for future work for the research community. The first is generic strategies that can be implemented based on deployment namely network architectures, virtualization environments, and cloud configurations. The second area of improvement is to combine the state of art deployment practices that are prototyped along with best practices of COAF. The last generic area of improvement identifies the enhancements that could be made for each of the modules of COAF.

7.1.1 Generic strategies for enhancing COAF

The current implementation of COAF is focused on testing and analysing the application behaviour on Multi-core platforms hosted within enterprise LAN even though COAF is not limited to any particular network considerations. This implementation can be enhanced to include the applications hosted on a wide area network, thereby capturing the results and behaviour of COAF for other network considerations such as network latency, various "Time to Live" session settings, transport protocols, and remote network exceptions. COAF currently uses Java based Axis2 framework and it can be enhanced or rewritten to support other popular operating environments such as Microsoft .Net. COAF uses "Solaris Zones and containers" for creating various core configurations and it would be interesting to study the effect of other explicit virtualization environments such as VMware. The pluggable nature of COAF can be combined with new advancements in cloud deployments [84], [229] to make powerful enterprise application hosting infrastructure on the cloud [234]. It would be interesting to study the performance improvements achieved by combining the bridging model [227] with portable algorithms and COAF.

7.1.2 Improvements on current "State of Art" deployments

There are Multi-core specific design approaches and research prototypes that can be augmented with the abilities of COAF system. These approaches include Multi-core specific operating system designs [22], [104], [154], [188], [240]; the workload assignments specific to asymmetric processor configurations [194]; , and cache (infrastructure) aware scheduling [76],

[129], [258]. The effect of these approaches can be further studied and analysed using the kernel observation methodologies and threshold configuration methodologies of COAF. There are design time approaches namely the ability of the application designer to identify and annotate the functional blocks [40]; and the ability to pass the application context to the operating system through hints [218] can be combined with administrator's knowledge in grouping and deploying the service requests for efficient processing. SOSOA execution engine [34] is a custom developed engine that uses similar approach as COAF in leveraging the hardware deployment configuration [33] for efficient processing at Multi-core. Similarly, a compiler based, cache topology aware code optimization scheme [132] further validates the insights provided by in-memory experiments under COAF. It would be interesting to blend the architecture framework such as Voltron [257], and the automatic parallelization libraries [134] such as SliCer [140] along with COAF to compare and understand the performance improvements additionally provided by COAF.

7.1.3 Improvements to the six Modules of COAF

Improvements are possible to the six modules of COAF to make it as a comprehensive enterprise framework for application and infrastructure deployment. The Resource definition in COAF is an implementation of web services resource framework. This implementation could be extended to include other standard models such as CIM model to represent various kinds of components used in the infrastructure with fine grained details (such as nport, eport, LUN number etc. for a SAN device). Behaviour Information module could be enhanced to include instrumentation frameworks other than DTrace. DTrace framework is comprehensively supported on Solaris but not on other popular operating systems. Hence, the extensive observations that we could make using DTrace is currently limited to Solaris based deployments. There are other evolving trace frameworks such as LTTNG can be plugged into COAF framework. Further these instrumentation frameworks could be standardized for popular infrastructures such as Apache, PostGreSQL and virtualized environments like VMware. Threshold Configuration module could be enhanced to include additional performance parameters apart from Cache miss for correlations against system configuration parameters. The correlation mechanisms can be extended to include other sophisticated correlation algorithms. Configuration server of the Core Configuration module could be made as "infrastructure deployment reference model" containing the predefined mapping for possible deployment

configuration that is used in the enterprise. This map then could be used for replacing an existing or new hardware and software. Solaris Zones and Containers concepts are extensively used to demonstrate the ability to bundle the cores in different core configurations. Further investigations and improvements can be made to include this ability for other operating systems.

7.1.4 Improvements to CCE and IPE

The parameters CCE and IPE are derived using Cache miss as the key parameter. The current derivation assumes equal (100 percent) weightage for all the variables. This equation is formulated as a methodology to quantitatively represent the impact of cache misses on the performance. The exact impact of cache misses could be formulated by varying the weightages for the variables for arriving at an enhanced formulation for CCE and IPE. Additionally, the other aspects of the process scheduling such as in-order, out-of-order etc. can be included as part of the formulation to enhance the usefulness of CCE and IPE.

REFERENCES

[1] Abdellatif, T., Kornaś, J., and Stefani, J.B. J2EE packaging, deployment and reconfiguration using a general component model. *Lecture Notes in Computer Science 3798*, 2005, 134-148. DOI: 10.1007/11590712_11.

[2] Adiga N.R. et al. - An overview of the Blue Gene/L supercomputer. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing.* (November 2002). pp. 1–22

[3] Aggarwal, N., Ranganathan, P., Jouppi, N.P., Smith, J.E., Configurable isolation: building high availability systems with commodity Multi-core processors, In *Proceedings of the ACM 34th annual international symposium on Computer architecture (ISCA '07)*, 35-2, May 2007, New York, NY, 470-481, doi>10.1145/1250662.1250720.

[4] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. *Web services: Concepts, architectures and applications*, 2004, Springer Verlag, ISBN 3-540-44008-9

[5] Altman, E., Arnold, M., Bordawekar, R., Delmonico, R.M., Mitchell, N., and Sweeney, P.F. Observations on tuning a Java enterprise application for performance and scalability, *IBM Journal of Research and Development*, 54-5, (Sept.-Oct.2010), 494-505, doi>10.1147/JRD.2010.2057090.

[6] Amazon.com Corporation. *Amazon web services*, 2011. http://aws.amazon.com/documentation/ (Last accessed:March 20, 2011)

[7] Amazon.com Corporation. *Worlds largest shopping site*, 2011. www.amazon.com (Last accessed: March 20, 2011).

[8] AMD Corporation. *Next generation Multi-core*, 2011 http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/Pages/technologies.aspx (Last accessed:March 20, 2011)

[9] Apache Software Foundation. *Apache Axis2.* http://axis.apache.org/axis2/java/core/ (Last accessed: March 20, 2011).

[10] Apache Software Foundation. *Apache Felix 3.0.9. Dynamic service deployment framework.* March 2011. http://felix.apache.org/site/index.html (Last accessed:March 20, 2011).

[11] Apache Software Foundation. *Apache Hadoop.* http://hadoop.apache.org/core/ (Last accessed:March 20, 2011).

[12] Apache Software Foundation. *Apache Web server*. http://www.apache.org/ (Last accessed:March 20, 2011).

[13] Apache Software Foundation. *JMeter:Performance testing framework*. http://jakarta.apache.org/jmeter/ (Last accessed:March 20, 2011).

[14] Apache Software Foundation. *MINA 2.0.2 Network Application framework*, December 2010. http://mina.apache.org/ (Last accessed:March 20, 2011).

[15] Apache Software Foundation. *Muse v2.2*, March 2007. http://ws.apache.org/muse/ (Last accessed:March 20, 2011).

[16] Apache Software Foundation. *PubScribe*, October 2005. http://svn.apache.org/repos/asf/web services/archive/pubscribe/site/index.html (Last accessed:March 20, 2011).

[17] Apache Software Foundation. *Servicemix v4.3.0*, March 2011. http://servicemix.apache.org/home.html (Last accessed:March 20, 2011).

[18] Apparao, P., and Bhat, M. A detailed look at the characteristics of XML parsing. In *Proceedings of Workshop on Building Block Engine Architectures for Computers and Networks, Held along with ASPLOS-XI (Beacon 04), October 2004.*

[19] Apparao, P., Iyer, R., Morin, R., and et al. Architectural characterization of an XML-centric commercial server workload. In *Proceedings of 33rd International Conference on Parallel Processing (ICPP 04)*, (August 2004). DOI: 10.1109/ICPP.2004.1327935.

[20] Apple Corporation. *Grand Central Dispatch: A better way to do Multi-core*. March 2011. http://images.apple.com/macosx/technology/docs/GrandCentral_TB_brief_20090903.pdf (Last accessed:March 20, 2011).

[21] Ask.com Corporation. *What's your question?*, 2011. http://www.ask.com (Last accessed:March 20, 2011).

[22] Bader, D. A., Kanade, V., and Madduri, K. SWARM: A Parallel Programming Framework for Multi-core Processors. In *Proceedings of the Parallel and Distributed Processing Symposium (IPDPS '07)* (June 2007), IEEE, pp. 1-8. DOI: 10.1109/IPDPS.2007.370681

[23] Ban, K., Chow, K., Lee, Y.F. et al., Java Application Server Optimization for Multi-core Systems, Intel Technology Paper, Sept 2009.

[24] Baresi, L., Nitto, E.D., and Ghezzi, C. Towards Open World Software. *Computer*, 39, October 2006. 36–43.

[25] Barroso, L. A., Gharachorloo, K., McNamara, and et al. Piranha: A Scalable Architecture Based on Single Chip Multiprocessing. In *Proceedings of the International Symposium on Computer Architecture (ISCA '00)*, (June 2000). pp. 282-293.

[26] Barroso, L.A., Gharachorloo, K., and Bugnion, E. Memory system characterization of commercial workloads - Computer Architecture. In *Proceedings of the 25th Annual International Symposium(ISCA '98)*, (July 1998), Barcelona , Spain. pp. 3-14. DOI: 10.1145/279358.279363.

[27] Behren, R. V., Condit, J., and Brewer, E. Why Events Are A Bad Idea (for high-concurrency servers). In *Proceedings of the 9th Workshop on Hot Topics in Operating System (HotOS IX '03)*, (2003), Lihue, Hawaii, USA.

[28] Bienia, C. *Benchmarking Modern Multiprocessors*. Ph.D. Thesis. Princeton University, January 2011.

[29] Blake, G., Dreslinski, R.G., and Mudge, T. A survey of Multi-core processors. *Signal Processing Magazine 26*, 6 (October 2009), 26-37. DOI:10.1109/MSP.2009.934110.

[30] Blockbuster Inc. *Movies on demand*. http://www.blockbuster.com (Last accessed:March 20, 2011).

[31] Bloomberg, J. Predicting the future of XML & web services. *Service orientation market trends report*. ZapThink, LLC. Jan 2004. Document ID: ZTR-WS110.

[32] Bolzoni, M.L., Calzarossa, M. C., Mapelli, P., and Serazzi, G. A package for the implementation of static workload models. In *Proceedings of the 1982 ACM SIGMETRICS conference on Measurement and modelling of computer systems(SIGMETRICS '82)* (1982), pp. 58-67. DOI: 10.1145/1035332.1035303.

[33] Bonetta, D., Peternier, A., Pautasso, C., and Binder, W. A Multi-core-aware Runtime Architecture for Scalable Service Composition. In *Proceedings of IEEE Asia-Pacific Services Computing Conference (APSCC '10)* (2010). IEEE. pp.83-90.

[34] Bonetta, D., Peternier, A., Pautasso, C., and Binder, W. Towards Scalable Service Composition on Multi-cores. In *Proceedings of the 2010 International Conference on On the move to meaningful internet systems (OTM '10)*, (2010). Springer-Verlag Berlin, Heidelberg.

[35] Bouziane, H.L., Perez, C., and Priol, T. Extending Software Component Models with the Master-Worker Paradigm. *Journal of Parallel Computing 36* 2-3, (February 2010) Elsevier Science Publishers B. V. Amsterdam, The Netherlands. 86-103. DOI:10.1016/j.parco.2009.12.012.

[36] Bower, F.A., Sorin, D.J., and Cox, L.P. The Impact of Dynamically Heterogeneous Multi-core Processors on Thread Scheduling. *IEEE Micro 28*, 3, (May 2008), 17-25. DOI: 10.1109/MM.2008.46.

[37] Boyd., D.J. A pragmatic approach to temporary payment card numbers. *International Journal of Electronic Security and Digital forensics 2*, 3, (July 2009), 253-268. DOI:10.1504/IJESDF.2009.027521

[38] Bray, J. Paoli, C. M. Sperberg-McQueen, and et al. Extensible Markup Language (XML) 1.0 (4th edition). *World Wide Web Consortium*, 2004. http://www.w3.org/TR/2006/REC-xml-20060816/.

[39] Breslau, L., Knightly, E.W., Shenker, S., and et al. Endpoint admission control: Architectural issues and performance. In *Proceedings of the conference on Applications, Technologies, architectures, and Protocols for Computer Communication (ACM SIGCOMM '00)*, (October 2000), Stockholm, Sweden, pp. 57-69. DOI: 10.1145/347059.347400.

[40] Burnim, J., Necula G., and Sen, K. Separating functional and parallel correctness using nondeterministic sequential specifications. In *Proceedings of the 2nd USENIX conference on Hot topics in parallelism* (HotPar'10). USENIX Association Berkeley, CA, USA. 2010.

[41] Cai, Z., Kumar, V., Cooper, B.F., and et. al. Utility Driven Proactive Management of Availability in Enterprise-Scale Information Flows. *Lecture Notes in Computer Science, 4290/2006*, 2006, 382-403.

[42] Carr, N.G. The end of corporate computing. *MIT Sloan Management Review Spring 46, 3*, 2005, 67-73.

[43] Cecil, R. The Power of DTrace. *Sun Microsystems, Inc*, March 2007. http://developers.sun.com/solaris/articles/power_of_dtrace.pdf (Last accessed:March 20, 2011).

[44] Chakravarty, M. M. T. Lazy Thread and Task Creation in Parallel Graph-Reduction. *Lecture Notes In Computer Science on Implementation of Functional Languages 1467*, 1997, Springer-Verlag, 231-249.

[45] Chao, L. Hyper-Threading Technology. *Intel Technology Journal 06*. 1 (February 14, 2002).

[46] Chazalet, A., and Lalanda, P. Deployment of Services Applications in Services Execution Environments. In *Proceedings of 33rd Annual IEEE International Computer Software and Applications Conference, (COMPSAC' 09)* (July 2009) pp.509-516.

[47] Chen, S., Ge, J., Tao, X., and Lu, J. A transaction model for context-aware applications. In *Proceedings of the 2nd international conference on Advances in grid and pervasive computing, (GPC'07)* (2007), Springer-Verlag Berlin, Heidelberg, pp.252-262,

[48] Chiu, K., and Lu, W. A compiler-based approach to schema specific xml parsing. In *Proceedings of First International Workshop on High Performance XML Processing*, 2004. http://www.cs.indiana.edu/~chiuk/pubs/chiu-ssp-sub.pdf (Last accessed:March 20, 2011).

[49] Chiu, K., Devadithya, T., Lu, W., and Slominski, A. A binary xml for scientific applications. In *Proceedings of the First International Conference on e-Science and Grid Computing (E-SCIENCE '05)* (2005), IEEE, pp.336-345

[50] Chow, C.K., Determination of Cache's capacity and its Matching Storage Hierarchy, IEEE Transactions on Computers, 25, 1976, 157-164.

[51] Cognizant Technology Solutions. *Building stronger businesses: Consulting, IT services, IT Infrastructure and BPO Services.* http://www.cognizant.com/

[52] Cox, M. Enterprise adoption of private clouds widespread and accelerating. *EChannelline Daily Channel News,* 7 October 2010. http://www.echannelline.com/usa/story.cfm?item=26212 (Last accessed:March 20, 2011).

[53] CPAN Project. *CPAN utilities.* http://www.cpan.org (Last accessed:March 20, 2011).

[54] Curbera, F., Leymann, F., Storey, T., Ferguson, D., and Weerawarana, S. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*, Prentice Hall, 2005.

[55] Dabek, F., Zeldovich, N., Kaashoek, F., and et al. Event-driven programming for robust software. In *Proceedings of the 10th ACM SIGOPS European Workshop (EW10 '02)*, (September 2002), pp.186-189. DOI: 10.1145/1133373.1133410.

[56] Dai, Z., Ni, N., and Zhu, J. A 1 cycle-per-byte XML parsing accelerator. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '10)*, (2010) ACM New York, NY, USA. pp. 199-208. doi:10.1145/1723112.1723148.

[57] David, S. M. EBay Creates Technology Architecture for the Future - *Patricia Seybold Group case study*, 2003. http://www.sun.com/service/about/success/recent/Sun_eBay6-2_forWeb.pdf (Last accessed:March 20, 2011).

[58] Dayal, U., Hsu, M., and Ladin, R. A Transactional Model for Long-Running Activities. In *Proceedings 17th International Conference on Very Large Data Bases (VLDB '91)*, (September 1991), Morgan Kaufmann Publishers Inc. San Francisco, CA, USA. pp. 113-122.

[59] Dean, J. and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM - 50th anniversary issue: 1958 - 2008 51*, 1, January 2008, 107-113.

[60] Dearle, A. Software Deployment, Past, Present and Future. In *Proceedings of Future of Software Engineering (FOSE '07)* (2007), IEEE Computer Society Washington, DC, USA pp. 269-284 DOI:10.1109/FOSE.2007.20.

[61] Dice, D. and Shavit, N. Understanding Tradeoffs in Software Transactional Memory. In *Proceedings of the international Symposium on Code Generation and Optimization (CGO '07)* (March 2007), IEEE Computer Society, Washington, DC, pp. 21-33. DOI: 10.1109/CGO.2007.38.

[62] Ding, J.J., Waheed, A., Yao, J., and Bhuyan, L.N. Performance characterization of multi-thread and Multi-core processors based XML application oriented networking systems. Original Research Article. *Journal of Parallel and Distributed Computing 70*, 5 (May 2010), 584-597.

[63] Draheim, D. *Business Process Technology: A Unified View on Business Processes, Workflows and enterprise applications.* 1st Edition, August 2010, Springer, 161-192.

[64] Dubey, A., and Wagle, D. Deliver Software as a Service, *Mckinsey Quarterly Web exclusive*, May 2007. http://www.mckinsey.de/downloads/publikation/mck_on_bt/2007/mobt_12_Delivering_Software_as_a_Service.pdf (Last accessed:March 20, 2011).

[65] Dustdar, S., and Schreiner, W. A survey on web services composition. *International Journal of Web and Grid Services 1*, 1, (August 2005), 1–30. DOI:10.1504/IJWGS.2005.007545.

[66] Dutta, S., and Franklin, M. Control Flow Prediction Schemes for Wide-Issue Superscalar Processors. *IEEE Transactions on Parallel and Distributed Systems 10*, 4, (April 1999), IEEE, 346-359. DOI: 10.1109/71.762815.

[67] EBay Inc. *Worlds largest online marketplace*, 2011. www.ebay.com (Last accessed:March 20, 2011).

[68] Ekanayake, J., Pallickara, S., and Fox, G. Map-Reduce for Data Intensive Scientific Analysis. In *Proceedings of the Fourth IEEE International Conference on eScience (ESCIENCE '08)*, (2008), pp.277-284. DOI: 10.1109/eScience.2008.59.

[69] Engelen, R.V. Constructing finite state automata for high performance XML web services. In *Proceedings of the International Symposium on Web Services (ISWS '04)*, (2004).

[70] Erenkrantz, R. J. *Computational REST: A new model for Decentralized, Internet-Sc-ale Applications*, University of California, Irvine, PhD Thesis, September 2009.

[71] Erl, T. *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[72] Etrade Corporation. *Cutting edge stock trading tools and platforms*, 2011. https://us.etrade.com/e/t/activetrading (Last accessed:March 20, 2011).

[73] ETrade corporation. *EquityEdge Online - The Easiest, Most Advanced Equity Compensation Solution Anywhere*, 2011. https://content.etrade.com/etrade/corpservices/equityedgeonlinefactsheet2009.pdf (Last accessed:March 20, 2011)

[74] Exertier, F. J2EE Deployment: the JOnAS case study. In *Proceedings of 1st Francophone Conference On Software Deployment and (Re)Configuration (DECOR'04)*, (October 2004), Grenoble, France, pp. 27-36.

[75] Facebook Corporation. *Wisdom of friends*, 2011. www.facebook.com (Last accessed:March 20, 2011).

[76] Fedorova, A., Seltzer, M., and Smith, M.D. Cache-fair thread scheduling for Multi-core processors. *Technical Report TR-17-06*, Harvard University, 2006.

[77] Fedorova, A., Seltzer, M., Small, C., and Nussbaum, D. Performance of Multithreaded Chip Multiprocessors and Implications for Operating System Design. In *Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC '05)*, (2005), USENIX Association, Berkeley, CA.

[78] Feo, J., Harper, D., Kahan, S., Koneeny, P. ELDORADO. In *Proceedings of the 2nd Conference on Computing Frontiers (CF '05), (Ischia, Italy)*, (May 2005). ACM, New York, NY, pp.28-34.

[79] Fielding, J. G., Mogul, J., Frystyk, and et. al. Hypertext transfer protocol: HTTP/1.1, *IETF*, June 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.txt (Last accessed:March 20, 2011).

[80] Fielding, R.T. *Architectural Styles and the Design of Network-based Software Architectures.* Ph.D Thesis, 2000.

[81] Flautner, K., Uhlig, R., Reinhardt, S., and Mudge, T. Thread-level parallelism and interactive performance of desktop applications. *In Proceedings of the Ninth International Conference on Architectural support for Programming Languages and Operating Systems (ASPLOS '00)* (November 2000), ACM New York, NY, USA. pp.129-138.

[82] Flissi, A., Dubus, J., Dolet, N., and Merle, P. Deploying on the Grid with DeployWare. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)* (2008) IEEE Computer Society Washington, DC, USA, pp.177-184. DOI:10.1109/CCGRID.2008.59

[83] Florian, D., and Barbara, P. Insights into Web Service Orchestration and Choreography, *International Journal of E-Business Research 2*, 1, (January-March 2006), Idea Group Publishing, 58-77.

[84] Fox, G.C.S, Bae, S.H., Ekanayakeb, J., Qiu, X., and Yuan, H. Parallel Data Mining from Multi-core to Cloudy Grids. In *Proceedings of the International Advanced Research Workshop on High Performance Computing and Grids (HPC '08)*, (2008).

http://grids.ucs.indiana.edu/ptliupages/publications/CetraroWriteupJan09_v12.pdf (Last accessed:March 20, 2011).

[85] Franklin, D.C., and Rosen., D. Electronic online commerce card with transaction proxy number for online transactions. *U.S. Patent No.5,883,810*, March 16, 1999

[86] Fruehe, J., Planning Considerations for Multi-core Processor Technology. Dell Whitepaper, May 2005

[87] Fruehe, J., Realizing Multi-Core Performance Advances in Dell PowerEdge Servers, Dell Whitepaper, Nov 2005.

[88] Fusion IO Inc. *Next generation flash memory*, 2011. www.fusionio.com (Last accessed:March 20, 2011).

[89] Galon, S., and Levy, M. Measuring Multi-core Performance. *Computer 41*, 11, (November 2008), 99-102.

[90] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994, Addison-Wesley, (ISBN 0-201-63361-2).

[91] Gara, A., Blumrich, M.A, Chen, D., et al. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development 49*, 2 (Mar 2005), IBM Corporation, Riverton, NJ, USA, 195-212.

[92] Gartner Inc. *Gartner Hype Cycle – Gartner Methodology.* http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp (Last accessed:March 20, 2011).

[93] Geer, D. Chip Makers Turn to Multi-core Processors. *Computer 38*, 5, (May 2005), IEEE, 11-13. DOI: 10.1109/MC.2005.160.

[94] Gold, B. T., Falsafi, G.B., Hoe, J.C., and Mai, K. REDAC: Distributed, Asynchronous Redundancy in Shared Memory Servers. *Technical Report*, (2008), Computer Architecture Lab at Carnegie Mellon, Pittsburgh, PA, USA.

[95] Goldstein, S., Schauser, K., and Culler, D. Enabling primitives for compiling parallel languages. In *Proceedings of Third Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, (May 1995), Rochester, NY.

[96] Golla, R. Niagara2: A Highly Threaded Server-on-a-Chip. *Sun Microsystems Slides*, August, 2006. https://wiki.cc.gatech.edu/Multi-core/images/8/89/Niagara.pdf (Last accessed:March 20, 2011).

[97] Goodchild, J. Multi-core processing infiltrates the enterprise. *Techtarget.com Article*. July 2006. http://searchwinit.techtarget.com/news/1197813/Multi-core-processing-infiltrates-the-enterprise (Last accessed:March 20, 2011).

[98] Google Inc. *Search Platform*, 2011. http://www.google.com/ (Last accessed:March 20, 2011).

[99] Gordon Blair, G., Coupaye, T., and Stefani, J.B. Component-based architecture: the Fractal initiative. *Annals of Telecommunications 64*, 1-2, (February 2009). 1-4, DOI: 10.1007/s12243-009-0086-1.

[100] Gottschalk, K., Graham, S., Kreger, H., and Snell, J. Introduction to web services architecture. *IBM Systems Journal 41*, 2, (2002), 170–177.

[101] Grahn, H. Transactional memory. *Journal of Parallel and Distributed Computing 70*, 10 (October 2010), Academic Press, Inc. Orlando, FL, USA, 993-1008.

[102] Greene, K. A new design for computer chips. *MIT Technology Review*, August 2007, http://www.technologyreview.in/business/19269/ (Last accessed:March 20, 2011).

[103] Gregg, B. DTrace Tools. *Wiki page, 2007.* http://www.brendangregg.com/dtrace.html (Last accessed:March 20, 2011).

[104] Gummaraju, J., Coburn, J., Turner, Y., and Rosenblum, M. Streamware: Programming general-purpose Multi-core processors using streams. In *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems (ASPLOS '08)*, (2008), pp. 297-307. DOI: 10.1145/1353536.1346319.

[105] Guo, T., and Li, G.Y. Neural data mining for credit card fraud detection. In *Proceedings of International Conference on Machine Learning and Cybernetics (MLC '08)*, (July 2008), pp. 3630-3634.

[106] Hammond, L., Hubbert, B., Siu, M., and et al. The Stanford Hydra CMP. *IEEE Micro 20*, 2, (March 2000), 71-84, DOI: 10.1109/40.848474.

[107] Hammond, L., Nayfeh, B., Olukotun, K. A single-chip multiprocessor, *Computer 30*, 9, (September 1997), 79-85. DOI: 10.1109/2.612253.

[108] Harizopoulos, S., and Ailamaki, A. A Case for Staged Database Systems. In *Proceedings of the First International Conference on Innovative Data Systems Research (CIDR '03)*, (January 2003).

[109] Harizopoulos, S., Shkapenyuk, V., and Ailamaki, A. QPipe: A Simultaneously Pipelined Relational Query Engine. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of data (SIGMOD '05)*, (2005), Baltimore, MD, pp.383-394. DOI:10.1145/1066157.1066201

[110] Hartstein, A., Srinivasan, V., Puzak, T.R., and Emma, P.G., On the Nature of Cache Miss Behavior: Is It √2 ? Journal of Instruction-Level Parallelism, 10, Jun 2008, 1-22.

[111] Herlihy M, and Moss J. E. B. Transactional Memory: Architectural Support for Lock-free Data Structures. *In Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA '93)*, (1993), San Diego, CA, ACM Press, NY, USA, 289-300. DOI: 10.1145/165123.165164.

[112] Herlihy, M, Luchangco, V., Moir, M., and Scherer, W. N. Software transactional memory for dynamic-sized data structures. *In Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing (PODC '03)*, (July 2003), ACM Press, NY, pp. 92-101. DOI: 10.1145/872035.872048.

[113] Holmes, D. W., Williams, J. R., & Tilke, P. G. An Events Based Algorithm for Distributing Concurrent Tasks on Multi-Core Architectures. *Computer Physics Communications*, (October 2009), 341-354. DOI: 10.1016/j.cpc.2009.10.009.

[114] HP Corporation. *HP OpenView: Enterprise Management Software.* http://www.managementsoftware.hp.com/ (Last accessed:March 20, 2011).

[115] Huang, Y., Slominski, A., Herath, C., and Gannon, D. WS-Messenger: A Web Services based Messaging System for Service-Oriented Grid Computing. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '06)*, (May 2006), IEEE, pp.166-173. DOI:10.1109/CCGRID.2006.109

[116] Hutsell, W. Solid State Disks in the Enterprise. *SNIA Summer Symposium*, 2008. http://www.snia.org/forums/sssi/knowledge/education/Solid_State_Disks_in_the_Enterprise.pdf (Last accessed:March 20, 2011).

[117] IBM Corporation. *System Z-series mainframes*, 2011. http://www-03.ibm.com/systems/z/ (Last accessed:March 20, 2011).

[118] IBM Corporation. *Tivoli Provisioning Manager 5.1.0.2 Documentation*, 2011. http://publib.boulder.ibm.com/infocenter/tivihelp/v16r1/index.jsp (Last accessed:March 20, 2011).

[119] IBM Corporation. *Websphere Datapower SOA appliance*, 2011. http://www-01.ibm.com/software/integration/datapower/ (Last accessed:March 20, 2011).

[120] Institute of eCommerce. *eLibrary-EPayment Links*, March 2011, http://euro.ecom.cmu.edu/resources/elibrary/epaylinks.shtml (Last accessed: March 20, 2011).

[121] Intel Corporation. *IntelLab: Terascale computing research vision*, 2011. http://techresearch.intel.com/ResearchAreaDetails.aspx?Id=27 (Last accessed:March 20, 2011).

[122] Intel Corporation. *Advancing Multi-Core Technology into the Tera-scale Era*, 2011. http://techresearch.intel.com/ProjectDetails.aspx?Id=151 (Last accessed:March 20, 2011).

[123] Intel Corporation. *Intel Parallel Studio Evaluation Guide: Optimize an Existing Program by Introducing Parallelism*, 2011. http://software.intel.com/sites/products/evaluation-guides/docs/intelparallelstudio-evaluationguide-add-parallelsim.pdf (Last accessed:March 20, 2011).

[124] Intel Corporation. *Threading Building Blocks: Scalable Programming for Multi-Core*, October 2010. http://software.intel.com/en-us/articles/intel-threading-building-blocks-scalable-programming-for-Multi-core/ (Last accessed:March 20, 2011).

[125] Intel Corporation. *VTune Amplifier XE 2011*. http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/ (Last accessed:March 20, 2011).

[126] Intuit Corporation. *Turbotax-Tax Preparation Software*, 2011. http://turbotax.intuit.com/ (Last accessed:March 20, 2011).

[127] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of European Conference on Computer Systems (EUROSYS '07)*, (March 2007), ACM, NY, pp.59-72. DOI: 10.1145/1272996.1273005.

[128] JCache project. *Open source cache solutions in Java*. http://java-source.net/open-source/cache-solutions/jcache (Last accessed:March 20, 2011).

[129] Jin, L., Lee, H., and Cho, S. A flexible data to L2 cache mapping approach for future Multi-core processors. In *Proceedings of the 2006 workshop on Memory system*

performance and correctness (MPSC '06) (2006), pp. 92-101. DOI: 10.1145/1178597.1178613.

[130] Kamboj, R., and Chen, J. Best practices for deploying MySQL on Solaris. *Presentation made at Products and services session in MySQL conference & Expo*, (April 2008), Santa Clara, CA, USA. http://www.scribd.com/doc/2602122/Best-Practices-for-Deploying-MySQL-on-the-Solaris-Platform-Presentation-1 (Last accessed:March 20, 2011).

[131] Kamp, P.H., and Watson R.N.M. Jails: Confining the omnipotent root. *The FreeBSD Project*. http://phk.freebsd.dk/pubs/sane2000-jail.pdf (Last accessed:March 20, 2011).

[132] Kandemir, M., Yemliha, T., Muralidhara, S., and et al. Cache topology aware computation mapping for Multi-cores. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming language design and implementation (PLDI '10)* (June 2010), pp.74-85. DOI:10.1145/1806596.1806605.

[133] Kaufmann, R., and Gayliard, B. Multi-core Processors, *Dr.Dobbs*. January 13, 2009. http://drdobbs.com/high-performance-computing/212900103 (Last accessed:March 20, 2011).

[134] Kim, H., and Bond, R. Multi-core software technologies. *IEEE Signal Processing Magazine 26*, 6 (November 2009), 80-89

[135] Kim, S., Chandra, D. and Solihin, Y. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture, In *IEEE Proceedings of the 13th International Conference of Parallel Architectures and Compilation Techniques (PACT 04)*, (2004), 111-122

[136] Knauerhase, R., Brett , P., Hohlt , B., Li , T., and Hahn S., Using OS observations to improve performance in Multi-core systems, IEEE Micro 38 3, 2008 54–66.

[137] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The Click modular router. *ACM Transactions on Computer Systems 18*, 3, (August 2000), 263–297.

[138] Kongetira, P., Aingaran, K., and Olukotun, K., Niagara: A 32-Way Multithreaded SPARC Processor. *IEEE Micro 25*, 2, (March 2005), 21-29. DOI: 10.1109/MM.2005.35.

[139] Krafzig, D., Banke, K., and Slama, D. *Enterprise SOA: Service-Oriented Architecture Best Practices.* Prentice Hall PTR, November 2004.

[140] Kwiatkowski, J., Iwaszyn, R. Automatic program parallelization for Multi-core processors. In *Proceedings of the 8th international conference on Parallel processing*

and applied mathematics: Part I (PPAM '09), (2010), Springer-Verlag Berlin, Heidelberg, pp. 236-245.

[141] Lam, T. C., Ding, J. J., and Liu J.C. XML Document Parsing: Operational and Performance characteristics. *Computer 41*, 9, (2008), 30-37. DOI:10.1109/MC.2008.403

[142] Larus, J., and Parkes, M. Using cohort scheduling to enhance server performance. *Technical Report MSR-TR-2001-39*, Microsoft Research, March 2001. http://research.microsoft.com/apps/pubs/default.aspx?id=69844 (Last accessed:March 20, 2011)

[143] Larus, J., Spending Moore's dividend, Communications of the ACM - Security in the Browser CACM, 52-5, May 2009, 62-69.

[144] Li, T., Lebeck, A.R., and Sorin, D.J. Spin Detection Hardware for Improved Management of Multithreaded Systems. *IEEE transactions on Parallel and Distributed Systems 17*, 6, (June 2006), 508 - 521. DOI: 10.1109/TPDS.2006.78.

[145] LinkedIn Inc. *Proessional network*, 2011. www.linkedin.com (Last accessed:March 20, 2011).

[146] Lowe, W. M., Noga, M. L., and Gaul, T. S. Foundations of fast communication via XML. *Annals of Software Engineering 13*, 1-4, (June 2002), 357-379.

[147] LTTng Project. *Linux Trace Toolkit, Comparison with SystemTap and DTrace.* http://lttng.org/content/comparison-systemtap-and-dtrace (Last accessed:March 20, 2011).

[148] Lttng project. *Linux Trace toolkit.* http://lttng.org/content/documentation (Last accessed:March 20, 2011).

[149] Lu, W., and Gannon, D., ParaXML. A Parallel XML Processing Model on the Multi-core CPUs , *Technical Report:TR662,* School of Informatics and Computing, Indiana University, April 2008. http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR662 (Last accessed:March 20, 2011).

[150] Luhn., H.P. Computer for Verifying Numbers. *U.S. Patent 2,950,048*, August 1960.

[151] LXC Linux Containers project. *Linux Containers chroot on Steroids.* http://lxc.sourceforge.net/ (Last accessed:March 20, 2011).

[152] Mann, J. *AMD to phase out single-core Athlons* June 2007. http://www.techspot.com/news/25848-amd-to-phase-out-single-core-athlons.html (Last accessed: March 20, 2011).

[153] Marino, M.D. L2-Cache Hierarchical Organizations for Multi-core Architectures. In *Frontiers of High Performance Computing and Networking (ISPA '06)*, (2006), Springer, Berlin, pp. 74-83.

[154] Márquez, A.L., Gil, C., Baños, R., and Gómez, J. Parallelism on Multi-core processors using Parallel.FX. *Advances in Engineering Software*, (November 2010). DOI:10.1016/j.advengsoft.2010.10.006.

[155] Mccool, M.D. Scalable Programming Models for Massively Multi-core Processors. In *Proceedings of the IEEE 9*, 5, (April 2008), 816-831. DOI:10.1109/JPROC.2008.917731.

[156] McDougall, R., Mauro, J., and Gregg, B. *Solaris performance and tools: DTrace and MDB Techniques for Solaris 10 and OpenSolaris*, 1st edition, Prentice Hall, July 2006.

[157] Memcached project. *A distributed memory object caching system.* http://memcached.org/ (Last accessed:March 20, 2011).

[158] Merrit, R. CPU designers debate Multi-core future at the International Solid State Circuits Conference. *EE Times*, Feb 2008. http://www.eetimes.com/electronics-news/4076123/CPU-designers-debate-Multi-core-future (Last accessed:March 20, 2011).

[159] Microsoft Corporation. *Deploying .NET Applications Lifecycle Guide*, December 2007. http://support.microsoft.com/kb/913507 (Last accessed:March 20, 2011).

[160] Microsoft Corporation. *Excel services.* http://office.microsoft.com/en-us/excel-help/about-statistical-analysis-tools-HP005203873.aspx (Last accessed:March 20, 2011).

[161] Microsoft Corporation. *Microsoft Live platform.* http://explore.live.com/home (Last accessed:March 20, 2011).

[162] Microsoft Corporation. *What is an Exchange Server email account - applies to Microsoft Office Outlook 2003.* http://office.microsoft.com/en-us/outlook-help/what-is-an-exchange-server-e-mail-account-HA001095504.aspx (Last accessed:March 20, 2011).

[163] Mohr, E., Kranx, D., and Halstead, R. Lazy task creation: A technique for increasing the granularity of parallel programs. *IEEE Transactions on Parallel and Distributed Systems 2*, 3, (Jul 1991), 264–280. DOI:10.1109/71.86103.

[164] MontaVista Software, LLC. *Beyond virtualization: The MontaVista Approach to Multi-core SoC Resource Allocation and Control*, 2011. http://mvista.com/download/Whitepaper-Beyond-Virtualization.pdf (Last accessed:March 20, 2011).

[165] MySpace Inc. *Leading social entertainment destination.* http://www.myspace.com/ (Last accessed:March 20, 2011).

[166] Nieplocha, J., Márquez, A., Feo, J. and et al. Evaluating the Potential of Multithreaded Platforms for Irregular Scientific Computations, In *Proceedings of the 4th Intl. Conference on Computing Frontiers (CF '07)*, (2007), ACM, NY, USA, pp. 47-58. DOI: 10.1145/1242531.1242541.

[167] Nowell, M., Vusirikala, V., and Hays, R. Overview of Requirements and Applications for 40 Gigabit and 100 Gigabit Ethernet. *Ethernetalliance-Version 1.0*, August 2007. http://www.csc.ohio-state.edu/~panda/788/papers/1i_Overview_and_Applications2.pdf (Last accessed:March 20, 2011).

[168] NVIDIA Corporation. *Programmable GPUs for High Performance computing.* http://research.nvidia.com/ (Last accessed:March 20, 2011).

[169] Olukotun, K., Nayfeh, B., Hammond, L., Wilson, K., and Chang, K. The case for a single-chip multiprocessor. *In Proceedings of the Seventh international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '96)*, (October 1996), ACM, NY, pp. 2-11. DOI:10.1145/237090.237140.

[170] OpenVz project. *Linux OpenVZ.* http://wiki.openvz.org (Last accessed:March 20, 2011).

[171] Oracle Corporation. *Java Blue Prints.* http://java.sun.com/blueprints/code/index.html (Last accessed:March 20, 2011).

[172] Oracle Corporation. *MySQL.* http://dev.MySQL.com/doc/refman/5.0/en/security.html

[173] Oracle Corporation. Petstore. http://java.sun.com/developer/releases/petstore/ (Last accessed:March 20, 2011).

[174] Oracle Corporation. *Solaris LDOMS.* http://docs.sun.com/source/820-4914-10/chapter1.html (Last accessed:March 20, 2011).

[175] Oracle Corporation. *System Administration Guide: Oracle Solaris Containers - Resource Management and Oracle Solaris Zones*, PartNo: 817–1592, 2010.

[176] Orfali, R., Harkey, D., and Edwards, J. *The essential client/server survival guide (2nd edition.)*, 1996, John Wiley & Sons, ISBN:0-471-15325-7.

[177] Organization for the Advancement of Structured Information Standards (OASIS). *Universal Description, Discovery and Integration (UDDI)*, February 2005. http://www.oasis-open.org/committees/uddi-spec/ (Last accessed:March 20, 2011).

[178] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Distributed Management (WSDM) v1.1*, August 2006. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm (Last accessed:March 20, 2011).

[179] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Resources Framework (WSRF). v1.2*, April 2006.

[180] Ousterhout, J.K. Why Threads Are A Bad Idea (for most purposes). *Presentation given at the 1996 Usenix Annual Technical Conference*, January 1996. http://www.stanford.edu/class/cs240/readings/threads-bad-usenix96.pdf (Last accessed:March 20, 2011).

[181] Pallickara, S., and Fox, F. NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids, In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware (Middleware '03)*, (2003), Springer-Verlag, pp. 41-61.

[182] Pan, Y., Lu, W., Zhang, Y., and Chiu, K. A static load-balancing scheme for parallel xml parsing on Multi-core CPUs. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, (May 2007), IEEE, pp. 351-362. DOI:10.1109/CCGRID.2007.14.

[183] Papadimitriou, S., and Sun, J. Disco: Distributed co-clustering with map-reduce. In *Proceedings on IEEE International conference on Data Mining (ICDM '08)*, (December 2008), IEEE, pp. 512-521. DOI: 10.1109/ICDM.2008.142.

[184] Papazoglou, M.P., and Georgakopoulos, D. Service-Oriented Computing, *Communications of the ACM* 46, 10, (October 2003), ACM, 25-28.

[185] Parallels Holding Ltd. *Top ten considerations for choosing a server virtualization technology:Parallels® Virtuozzo Containers*, 2010. http://www.smbvirtualization.net/files/2010/10/top_ten_considerations_for_choosing_server_virtualization_technology.pdf (Last accessed:March 20, 2011).

[186] Patterson, D. The trouble with Multi-core. Chipmakers are busy designing microprocessors that most programmers can't handle, *IEEE Spectrum*, July 2010.

[187] Peltz, C. Web Services Orchestration and Choreography. *Computer 36*, 10, (October 2003), IEEE, 46-52.

[188] Peter, S., Schüpbach, A., Barhamy, P., and et al. Design Principles for End-to-End Multi-core Schedulers. In *Proceedings of the 2nd USENIX conference on Hot topics in parallelism (HotPar'10)* (2010), USENIX Association Berkeley, CA, USA.

[189] Ramkrishna R. B., and Fisher A. J. *Instruction Level Parallelism, Encyclopedia of Computer Science*, 4th Edition, 2003, John Wiley and Sons Ltd, 883-887.

[190] Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G. R., and Kozyrakis, C. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA '07)*, (2007), pp. 13-24. DOI: 10.1109/HPCA.2007.346181.

[191] Red Hat Inc. *JBossWS v3.4.1-Web service framework for JBoss AS*, Jan 2011. http://community.jboss.org/wiki/JBossWS (Last accessed:March 20, 2011).

[192] Roetter, A. Writing multithreaded Java applications-Learn to avoid problems common in concurrent programming. *IBM Developerworks Journal*, February 2001, http://www.ibm.com/developerworks/library/j-thread.html (Last accessed:March 20, 2011).

[193] Ruggaber, R. Internet of Services - A SAP Research Vision. In *Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '07)* (2007), IEEE Computer Society, DOI: 10.1109/WETICE.2007.152.

[194] Saez, J.C., Prieto, M., Fedorova, A., and Blagodurov, S. A comprehensive scheduler for asymmetric Multi-core systems. In *Proceedings of the 5th European conference on Computer systems (EuroSys '10)* (2010), pp. 139-152. DOI: 10.1145/1755913.1755929.

[195] Salesforce.com Corporation. *CRM:The leader in customer relationship management*, 2011. http://www.salesforce.com (Last accessed:March 20, 2011).

[196] SAP Corporation. *SAP Business Management software solutions, applications, and services*, 2011. http://www.sap.com (Last accessed:March 20, 2011).

[197] SAP Corporation. *SAP Netweaver :Adaptive technology for the Networked Enterprise*, 2011. http://www.sap.com/platform/netweaver/index.epx (Last accessed:March 20, 2011).

[198] Schroth, C., and Janner, T. Web 2.0 and SOA. Converging concepts enabling the Internet of services. *IT Pro IEEE Computer Society*, May 2007. http://www.computer.org/portal/web/buildyourcareer/fa008 (Last accessed:March 20, 2011).

[199] Searls, R. JSR-000088:JavaTM 2 Enterprise Edition Deployment API Specification, Version 1.1 J2EE Application Deployment. *Oracle Sun Network*, November 2003.

[200] Serena Corporation. *Serena Mashup composer:A revolution in Application development*, 2011. http://help.serena.com/mashups/2009R1/sbms_composer_saas_guide.pdf (Last accessed:March 20, 2011)

[201] Shen, K., Request behavior variations., *In Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, (Mar 2010), 103-116.

[202] Sholler, D. SOA User Survey: Adoption Trends and Characteristics. *Gartner Report*. 2008. ID Number: G00161125. http://www.gartner.com/DisplayDocument?id=765720 (Last accessed:March 20, 2011).

[203] Shoup, R., and Pritchett, D. The eBay architecture-Striking the balance between site stability, future velocity, performance and cost, *SD Forum 2006*, November 29, 2006.

[204] Siwiki Project. *DTrace Topics Guide. Solaris Internals Wiki*, March 2011. http://www.solarisinternals.com/wiki/index.php/DTrace_Topics_Guide (Last accessed:March 20, 2011).

[205] Slater, P., Hill, R., and Hogg, J. Deploying .NET Framework-based Applications. *Microsoft Patterns and practices*, MSDN Journal Library, June 2003. http://www.microsoft.com/downloads/en/details.aspx?FamilyId=5B7C6E2D-D03F-4B19-9025-6B87E6AE0DA6&displaylang=en (Last accessed:March 20, 2011).

[206] Snir, M., and Yu, J. On the Theory of Spatial and Temporal Locality. *Technical Report No.UIUCDCS-R-2005-2611*, Department of Computer Science, UIUC, July 2005.

[207] Song, Y., Sailer, A., and Shaikh, H. Problem classification method to enhance the ITIL incident and problem. *In the Proceedings of the 11th IFIP/IEEE international conference*

on Symposium on Integrated Network Management.(IM '09). (June 2009), IEEE Press Piscataway, NJ, USA, pp-295-298.

[208] Stackless.com Inc. *Stackless Pyton.* http://www.stackless.com/ (Last accessed:March 20, 2011).

[209] Stallings, W. (2004). *Operating Systems Internals and Design Principles.* (Fifth International Edition), Prentice Hall, ISBN 0-13-147954-7, 405-491.

[210] Staten, J. Your Thoughts: How mature are cloud computing services? *Forrester Research*, June 2009. http://www.zdnet.com/blog/forrester/your-thoughts-how-mature-are-cloud-computing-services/227 (Last accessed:March 20, 2011).

[211] Stephen, D. H. Worldwide SOA-Driven Software 2009-2013 Forecast. *IDC Report*, Aug 2009. Doc #219327. http://www.marketresearch.com/product/display.asp?productid=2424614&xs=r (Last accessed:March 20, 2011).

[212] Stokes, J. *Inside the machine: an illustrated introduction to microprocessors and Computer Architecture.* No Starch Press, December 2006, 220-221.

[213] Sulaiman, D.R., Hardware Based: Dynamic Branch Prediction for Microprocessors Energy Reduction in Portable Systems. *International Journal of Engineering Studies 2,* 2, (2010), Research India Publications, 223–235. http://www.ripublication.com/ijes/ijesv2n2_10.pdf (Last accessed:March 20, 2011).

[214] Sun, C., He, L., Wang, Q., and Willenborg, R. Simplifying Service Deployment with Virtual Appliances. In *Proceedings of IEEE International Conference on Services Computing (SCC '08),* (2008), pp.265-272. DOI:10.1109/SCC.2008.53

[215] Symantec Corporation. *Norton safe web from Symantec*, 2011. http://safeweb.norton.com/ (Last accessed:March 20, 2011)

[216] SystemTap project. *SystemTap Linux monitoring.* http://sourceware.org/systemtap/documentation.html (Last accessed:March 20, 2011).

[217] Tallent, N. R. and Crummey, J. M. M. Effective performance measurement and analysis of multithreaded applications. In *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '09),*(February 2009), ACM, NY, pp.229-240. DOI: 10.1145/1504176.1504210.

[218] Taura, K., and Yonezawa, A. Fine-grain multithreading with minimal compiler support: A cost effective approach to implementing efficient multithreading languages. In *Proceedings of the 1997 ACM SIGPLAN Conference on Programming Language Design and Implementation(PLDI '97)*, (June 1997), ACM, NY, USA, pp.320-333. DOI: 10.1145/258915.258944.

[219] Teradata Corporation. *Data appliance and business intelligence.* www.teradata.com (Last accessed:March 20, 2011).

[220] Terracotta Inc. *Distributed Shared Object platform.* http://www.terracotta.org/enterprise-suite/ (Last accessed:March 20, 2011).

[221] Tilera Corporation. *Manycore without boundaries.* http://www.tilera.com/technology (Last accessed:March 20, 2011).

[222] Tilera Corporation. *Manycore without boundaries-Multi-core Development Environment(MDE).* March 2011. http://www.tilera.com/development_tools (Last accessed:March 20, 2011).

[223] Toupin, D. Using Tracing to Diagnose or Monitor Systems. *IEEE Software 28*, 1, (Jan 2011), 87-91. DOI:10.1109/MS.2011.20.

[224] Tseng, J.H., Yu, H., Nagar, S., Dubey, N., Franke, H., and Pattnaik, P., Performance Studies of Commercial Workloads on a Multi-core System, in *IEEE 10th International Symposium on Workload Characterization (IISWC 2007)*, Sept. 2007, 57-65.

[225] Tullsen, D., Eggers, S., and Levy, H. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of 25 Years of the International Symposia on Computer Architecture – Selected Papers (ISCA '98).* (July 1998), ACM, NY. pp. 533-544. DOI: 10.1145/285930.286011.

[226] Twitter Inc. *Real time messaging.* www.twitter.com (Last accessed:March 20, 2011).

[227] Valiant, L.G. A bridging model for Multi-core computing. *Proceedings of the 16th annual European symposium on Algorithms (ESA '08)* (2008), pp.154-166. DOI: 10.1007/978-3-540-87744-8_2.

[228] Valipour, M.H., Amirzafari, B., Maleki, K.N., and Daneshpour, N. A Brief Survey of Software Architecture Concepts and Service Oriented Architecture. *In Proceedings of 2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT '09).* (Aug 2009), pp.34-38. DOI: 10.1109/ICCSIT.2009.5235004.

[229] Vaquero, L.M., Merino, L.P., and Buyya, R. Dynamically Scaling Applications in the Cloud. *ACM SIGCOMM Computer Communication Review 41*, 1 (January 2011), 45-52. DOI: 10.1145/1925861.1925869.

[230] Vendavo Corporation. Price optimization and price management software. http://www.vendavo.com (Last accessed:March 20, 2011).

[231] Venugopal, S., and Ganesan, K. Tools for observing kernel behaviour. In *Proceedings of National Conference on Recent Trends in Information Technology (NCRTIT '07)* (August 2007), Chennai, India.

[232] Venugopal, S., and Desikan, S.K. Modernization challenges in transition to the HD world. *Lecture notes in Broadcast Engineering Society*, Mar 2010.

[233] Venugopal, S., Desikan, S.K. and Ganesan, K. Connection Oriented Framework for Effectively Using Multicore in the Enterprise. In *Proceedings of 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS '11)* (February 2011), IEEE. Paris, DOI:10.1109/NTMS.2011.5720651.

[234] Venugopal, S., Desikan, S.K. and Ganesan, K. Effective Migration of Enterprise Applications in Multicore Cloud. In *Proceedings of International Workshop on Cloud Computing & Future of work (UCC '11)* (December 2011), IEEE, Melbourne, 463-468, ISBN: 978-0-7695-4592-9.

[235] Venugopal, S., Desikan, S.K. and Ganesan, K. Connection Oriented Framework for enterprise applications. *International Journal of Advanced Computing. Volume 4, Issue 2, Dec 2012* Accepted for publishing.

[236] Virident Inc. High performance large memory. www.virident.com (Last accessed:March 20, 2011).

[237] Viswanathan, M., Shaikh, H., Sailer, A., and et al. ERMIS: Designing, developing, and delivering a remote managed infrastructure services solution. *IBM Journal of Research and Development 53*, 6, (November 2009), IBM Corporation, Riverton, NJ, USA, 871-888. DOI: 10.1147/JRD.2009.5429036.

[238] VMWare Corporation. *Virtualization solution for servers.* http://www.vmware.com (Last accessed:March 20, 2011).

[239] Welsh, M., Culler, D., and Brewer, E. SEDA: Architecture for well conditioned, scalable Internet services. In *Proceedings of the 18th ACM Symposium on Operating Systems*

Principles (SOSP '01). (October 2001), ACM, NY, USA, pp.230-243. DOI: 10.1145/502034.502057.

[240] Wentzlaff, D., and Agarwal, A. Factored operating systems (fos): the case for a scalable operating system for Multi-cores. *ACM SIGOPS Operating Systems Review 43*, 2, (April 2009), 76–85. DOI: 10.1145/1531793.1531805.

[241] Weske, M. *Business Process Management: Concepts, Languages, and Architectures*. Springer, November 2007.

[242] Wikipedia project. *Web service frameworks*, 2011. http://en.wikipedia.org/wiki/List_of_web_service_frameworks (Last accessed:March 20, 2011).

[243] Wood, T., Gerber, A., Ramakrishnan, K.K., Shenoy, P., and Merwe, J. V. The Case for Enterprise-Ready Virtual Private Clouds. In *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '09)* (June 2009), USENIX Association Berkeley, CA, USA.

[244] World Wide Web Consortium (W3C). *Extensible markup language (XML)*. http://www.w3.org/XML (Last accessed:March 20, 2011).

[245] World Wide Web Consortium (W3C). *Simple Object Access Protocol (SOAP) v1.2*, Arpil 2007. http://www.w3.org/TR/soap12/ (Last accessed:March 20, 2011).

[246] World Wide Web Consortium (W3C). *Web Service Architecture*. http://www.w3.org/TR/ws-arch/. (Last accessed:March 20, 2011).

[247] World Wide Web Consortium (W3C). *Web Service Description Language (WSDL) v2.0*, June 2007. http://www.w3.org/TR/wsdl20/ (Last accessed:March 20, 2011).

[248] World Wide Web Consortium (W3C). Web services framework. *W3C Workshop on Web Services*, April 2001, San Jose, CA, USA. http://www.w3.org/2001/03/WSWS-popa/paper51 (Last accessed:March 20, 2011).

[249] World Wide Web Consortium (W3C). *XML binary characterization properties*. http://www.w3.org/TR/xbc-properties/. (Last accessed:March 20, 2011).

[250] XAMPP Project. *XLAMP stack*. http://sourceforge.net/projects/xampp/ (Last accessed:March 20, 2011).

[251] Xen Project. *Xen hypervisor*. http://www.xen.org/products/xenhyp.html (Last accessed:March 20, 2011)

[252] Xiaoya, X., Bao B., Ding. C., and Shen, K., Cache Conscious Task Regrouping on Multi-core Processors, in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, (2012), 603-611.

[253] Yahoo Inc. *Web portal, search and directory*. http://www.yahoo.com/ (Last accessed:March 20, 2011).

[254] Yu, J., Baghsorkhi, S., and Snir, M. A New Locality Metric and Case Studies for HPCS Benchmarks. *Technical Report, No. UIUCDCS-R-2005-2564*, Department of Computer Science, UIUC, April 2005.

[255] Zhang, E.Z., Jiang, Y., and Shen, X. Does cache sharing on modern CMP matter to the performance of contemporary multithreaded programs? In *Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP '10)*. (May 2010), ACM, NY, USA, pp.203-212.

[256] Zhao, Li., and Bhuyan, L. Performance Evaluation and Acceleration for XML Data Parsing. In *9th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW '06) (2006)*, Austin, Texas.

[257] Zhong, H. Architectural and compiler mechanisms for accelerating single thread applications on Multi-core processors. *Doctoral Dissertation*, University of Michigan, Ann Arbor, MI, USA 2008.

[258] Zhuravlev, S., Blagodurov, S., and Fedorova, A. Addressing shared resource contention in Multi-core processors via scheduling. SIGPLAN Not. In *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems (ASPLOS '10)* (March 2010), pp. 129-142.

[259] Ziegler., J. *Everything you ever wanted to know about CC's*, http://euro.ecom.cmu.edu/resources/elibrary/everycc.htm (Last accessed: March 20, 2011).

APPENDIX I

System Probe Tools and their Comparison : DTrace / Truss / SystemTap

Appendix I deal with the comparison of standard probes that exists currently for observing and logging kernel level parameters. There are four state of art tools evaluated, namely SystemTap, LTTng, DTrace and kprobes. After preliminary tests, the kprobes were determined to be timing sensitive and hence it was discarded from further analysis in the context our prototype test bed. The comprehensive analysis for the remaining three tools described below is available in the literature [147].

Linux Tools – 1 – LTTng (Linux Trace Toolkit Next Generation) is a tracer being actively developed on Linux by IBM. This tool is a kernel patch accompanied by a tool chain (ltt-control) to control the tracing, as well as a trace viewing and analysis program (LTTV). LTTng includes a set of kernel instrumentation points useful for debugging a wide range of bugs that are otherwise extremely challenging. These include performance problems on parallel systems and on real-time systems. Custom instrumentation is easy to add in LTTng. LTTng is designed for minimal performance impact and has a near zero impact when not tracing.

Linux Tools – 2 – SystemTap is another system monitoring tool in Linux. It can dynamically instrument the kernel and provide enough data to analyze the performance problems. It provides information from the sub system level to routine level granularity. SystemTap instrumentation incurs low load when enabled and zero load when disabled. SystemTap also provides facilities to define instrumentation points in a high level language and to aggregate and analyze the instrumentation data.

DTrace – Solaris – DTrace is a standard production grade Solaris tool that is used to analyse system calls and signals, and more recently has been enhanced to follow user level function calls. DTrace provides over 30,000 probes, and has the capability to drill down to each instruction of a process. Truss by default only looks at system calls and signals. Also, sTrace and Truss are invasive in nature and they interfere when the system is under observation. The following table lists the criteria that are used for selection of the tool.

Project	LTTng	SystemTap	DTrace
operating system support	Linux	Linux	Solaris, Mac OS X, BSD, QNX
license	Kernel tracer: GPLv2	GPL	CDDL
development began	Jan-05	Jan-05	Oct-01
development status	Ongoing	Ongoing	stable with continuing development
Collaborators	Multi-Core Association Linux Foundation	RedHat, IBM and Oracle	Oracle (Sun)
Language style	C	Scripting	Scripting
Speculative tracing	work in progress	yes	Yes
Probe execution	optimized native code	optimized native code	interpreted bytecodes
concurrent probes on multiprocessors	yes	Yes	Yes
trace script language programs	yes, (script language calls C)	not yet	yes: Ruby, JavaScript, Perl, Python, PHP, APL, Bourne shell, ksh, zsh, Tcl
timer-based probing	no	Yes	Yes
Analysis performed	offline (post-mortem)	Online	Online

Non invasive probing is essential as it does not affect the functioning of the key processes when they are running on the core. The probes needed to be online to reflect and provide any feedback. Both System Tap and DTrace provide this facility . The next important criteria were the stability of the probes themselves, as well as their ability to be scripted easily. DTrace scored in both these conditions. The execution of the probe by directly involving interpretation of byte code, meant that no code had to be specifically recompiled. This is a key as enterprises view recompilation with suspicion and as a risk. The timer-based probing enables us to reconstruct the events that happened in a chronological sequence and DTrace and SystemTap scored in these. The following table presents the status of the tools at the time of the experiments.

Based on the above, DTrace was chosen as the new comprehensive system analysis tool that is less intrusive than other tools, and is safe to run in production.

APPENDIX II

LIST OF DTRACE PROBES TO CHOOSE

DTrace at the time of experiments had over 70,000 probes, the instrumentation point from which DTrace can collect data [204] [43]. Brendan Gregg's tool kit [103] ran over 200 special scripts. Probes consist of four-tuples as described below.

Tuple	Description
Provider	A library of related probes.
Module	The module the function belongs to, either a kernel module or user segment.
Function	The function name that contains the probe.
Name	The name of the probe.

Each DTrace probe writes a log into a corresponding log file. Each of these DTrace log files determine which data to be observed. Example, which process are running on which core, the behaviour of operating system scheduler, pid generation process and the processing queue etc. DTrace provides 19 pre built scripts to monitor the system at operating system level. These are

1 anonpgpid.d,
2 cputypes.d,
3 cpuwalk.d,
4 cpuxcallsbypid.d,
5 dispqlen.d,
6 intbycpu.d,
7 intoncpu.d,
8 inttimes.d,
9 loads.d,
10 minfbypid.d,
11 minfbyproc.d,
12 runocc.d,
13 swapinfo.d,
14 syscallbypid.d,
15 syscallbyproc.d,
16 vmbypid.d,
17 vmstat.d,
18 vmstat-p.d,
19 xvmstat.d

Fifteen of these scripts were used as probes and their respective description and data collected is presented in the following table.

List of 15 DTrace Probes and the probe headers used during Setup

No	Probe Name	Probe Class	Brief Description	Probe Headers
1	cpu_xcallsbypid	CPU	List the inter-processor cross-calls by process id. The inter-process cross calls is an indicator how much work a CPU sends to another CPU	PID,CMD,XCALLS
2	dispqlen_by_cpu	CPU	Prints the dispatcher Queue length	CPU ID, VALUE, DISTRIBUTION, COUNT
3	interrupt_by_cpu	CPU	prints the number of interrupts by CPU	CPU, INTERRUPTS
4	interrupt_time	CPU	lists the interrupt activity by device	DEVICE, TIME (ns)
5	minf_by_process	Memory	This program prints a report of minor faults by Process.	PID, CMD, MINFAULTS
6	minf_by_pid	Memory	This program prints a report of minor faults by PID. This script is to used to help determine which process was consuming the most memory and had cache faults during the sample.	PID, MINFAULTS
7	readb_by_process	Process	how many bytes are read by process	PID, CMD, BYTES
8	sample_process	Process		PID, SAMPLES , ARGS
9	signal_count	Counters	Summary of the different signals that occur for a process.	FROM, SIG, TO, COUNT
10	syscall_by_process	Process	system calls by process or by PID, process ID, process name, syscall name and number of syscalls for this PID	PID, CMD, SYSCALL, COUNT
11	syscall_count	Counters	Count of all syscalls	SYSCALL, COUNT
12	syscall_errors	Process	Collects the system errors that have occurred for a process.	PID, CMD, SYSCALL, ERRNO, COUNT
13	sysinfo_by_process	Process	Analyses process activity, bytes written or read by process, files opened by process	PID, CMD, STATISTIC, COUNT
14	vminfo_by_process	Process	how much of VM is consumed by process	PID, CMD, STATISTIC, VALUE
15	writeb_by_process	Process	how many bytes are written by process	PID, CMD, BYTES

LIMITATION OF DTRACE

There is a dedicated shared memory space provided for logging and creation in the DTrace framework. Logs stored onto the hard disk after DTrace has performed its non-invasive probing. DTrace best practices for a set log buffer size, from the general recommendation found in literature [156][130][217]. The time for logging should not affect the other processes with its buffer overflow in I/O while recording the logs. The prescribed limit in literature is [156][130][217] about 30 seconds, per probe, for a generic log file to write into memory.

In the prototype test bed - multiple probes are used simultaneously. Four probes will share the memory space to write the buffer. The 30 second timeslot window, is therefore, recomputed. The logs should have been captured for 30/4 = 7.5 seconds. By comparing the size of the headers and their respective output, as listed in the table above, and by observing that not all logs are of same size, the test bed ran the four DTrace probes safely for a 10 seconds time window within the buffer write limit.

APPENDIX III

STEPS TO CREATE SOLARIS ZONES

With recent advances in architecture of Niagara Virtualization in Solaris done using zones. As DTrace tracing does not occur across zones, we set up the instance of DTrace in one zone. Setting up Solaris Zones requires two prominent steps, (1) creating and assigning cores to a zone and (2) assigning the network properties for each zone. The following nine steps are detailed out below.

1. Assigning a single dedicate core for zone : add dedicated-cpu
2. To modify the assigned cores for zone : set ncpus=1-2
3. Assigning system privileges to zone : set limitpriv="default,sys_time"
4. Scheduling to FSS : set scheduling-class=FSS
5. Adding memory cap to zone : add capped-memory
6. Assigning 2G memory to zone : set physical=2048m
7. Add file system to the zone : add fs
8. Add mountpoint for zone : set dir=/usr/local
9. Connect zone mountpoint to global fs : set special=/opt/zones/research-lab1/local

We used the following four steps to assign network properties:

1. set IP type : set ip-type=exclusive
2. Add network device : add net
3. Set IP address : set address=192.168.234.10
4. Attach IP to device : set physical=hme0

APPENDIX IV

LOG COLLECTION FROM DTRACE

This appendix discusses how the logs of DTrace are used to reconstruct what is happening at each core. A process is a command in execution and in DTrace referred to by command (CMD) or process. In the DTrace framework, three probes determine how the characteristics of the core behaviour (1) dispqlen_by_cpu, (2) syscall_by_process and (3) minf_by_process. The fourth interrupt_by_cpu logs how cores behave with reference to each other and how many cores are used. dispqlen_by_cpu gives the volume of load on the CPU without any reference to the process currently executed. syscall_by_process refers to the number of system calls that are made by a process in focus. syscall_by_process also presents number of context switches that are made when the command is in execution. minf_by_process determines the number of faults that a process makes while executing. Together, syscall_by_process and minf_by_process logs present a picture of how a particular application is running and performing, and when and why the context is getting switched – while indirectly referencing the load based on queue length.

The log management and reconstruction process is done in four steps:

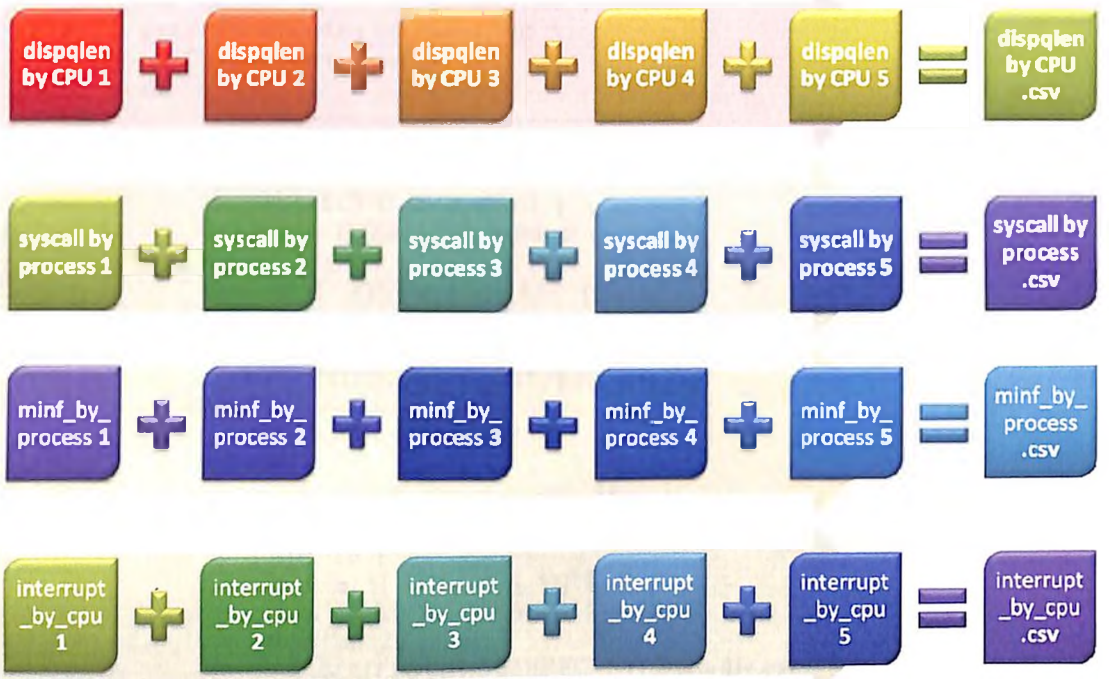
1) Log Preparation– Plan the buffer size, the number of probes and the time slots
2) Aggregation - Collecting data from four probes in these time slots
3) Filtering – Extraction and Group of desired processes (detailed in Appendix V).
4) Analysis – Recreating the Chronology (detailed in Appendix V).

Log preparation

The first part of the reconstruction is planning a timeslot window within which DTrace will observe the system. The number of operational cores is set to 9 with methodology as described in Appendix III. The Nine Cores are numbered - CPU 0, CPU 1, ... CPU 8. In the test bed, for the 9 cores configuration environment discussed, five time slots were taken within a 151 second time windows based on the log buffer size limitations *viz.* TS1 = 18:40:50 - 18:41:00, TS2 = 18:41:21 - 18:41:31, TS3 = 18:41:55 - 18:42:05, TS4 = 18:42:29 - 18:42:39, and TS5 = 18:43:11 - 18:43:21.

Log Aggregation

The output from the four probes behaviour - dispqlen_by_cpu, syscall_by_process, minf_by_process and interrupt_by_cpu is recorded respectively as dispqlen_by_cpu.1, syscall_by_process.1, minf_by_process.1, interrupt_by_cpu.1, dispqlen_by_cpu.2, syscall_by_process.2, minf_by_process.2, interrupt_by_cpu.2, ... dispqlen_by_cpu.5, syscall_by_process.5, minf_by_process.5 and interrupt_by_cpu.5 respectively. The next step consolidates these disparate log files into dispqlen_by_cpu.csv, syscall_by_process.csv, minf_by_process.csv and interrupt_by_cpu.csv as shown in the figure below.



Thus, the DTrace file processor is the aggregated based on the file type with the respective aggregator as shown in the figure above. This pluggable design indicated helps us to unify the processor scripts depend on the input type. The following script shown as a snapshot cleans the headers from the dispqlen_by_cpu.1 ... 5 and aggregates the required records into the file dispqlen_by_cpu.csv.

```
#!/bin/bash
export COUNT="$1"
export OUTPUTDIR="$2"

#
#########################################################
for DIRECTRY in `ls -1 | grep -v final`
do
#    ------------------------------------------
#    first make the directories at destination
#    then replicate sub-folders
#    ------------------------------------------
   echo $DIRECTRY
     mkdir -p $OUTPUTDIR/$DIRECTRY
     echo "$OUTPUTDIR/$DIRECTRY created"
#    ------------------------------------------
     for datafile in `ls -1 $DIRECTRY | \
     grep -iv dispqlen_by_cpu | cut \
     -d"." -f1 | sort | uniq`
     do
#          ------------------------------------------
          echo $DIRECTRY/$datafile
#          ------------------------------------------
          cat $DIRECTRY/$datafile.1 | \
          grep -v  CTSINGTOSUN04 | \
          head -1 | \
          sed 's/^[ \t]*//;s/[ \t]*$//' | \
          awk '{ $1=$1; print }' | \
          sed 's/ /,/g' \
        > $OUTPUTDIR/$DIRECTRY/$datafile.csv
#    ------------------------------------------
          for fileord in {1..$COUNT}
          do
               cat $DIRECTRY/$datafile.$fileord | \
               grep -v  CTSINGTOSUN04 | \
               sed 1d | \
               sed 's/^[ \t]*//;s/[ \t]*$//' | \
               awk '{ $1=$1; print }' | \
               sed 's/ /,/g' \
               >> $OUTPUTDIR/$DIRECTRY/$datafile.csv
          done
#          ------------------------------------------
#    ------------------------------------------
     done
#    ------------------------------------------
     echo finished $DIRECTRY
#    ------------------------------------------
done
```

Script - Code Snapshot of File Aggregator

APPENDIX V

RELEVANT DATA RECONSTRUCTION FROM DTRACE

This portion of appendix discusses how, from the aggregated data, we can reconstruct the chronology of what happened in the system. The syscall_by_process log returns the following records, as evidenced by the header – viz. PID, CMD, SYSCALL, and COUNT.

The PID represents the Process ID of a process and the CMD represents that Command that is executed. For example, the first two records in syscall_by_process.csv are

PID	CMD	SYSCALL	COUNT
1240	webseald	pollsys	1
1242	sendmail	pollsys	1

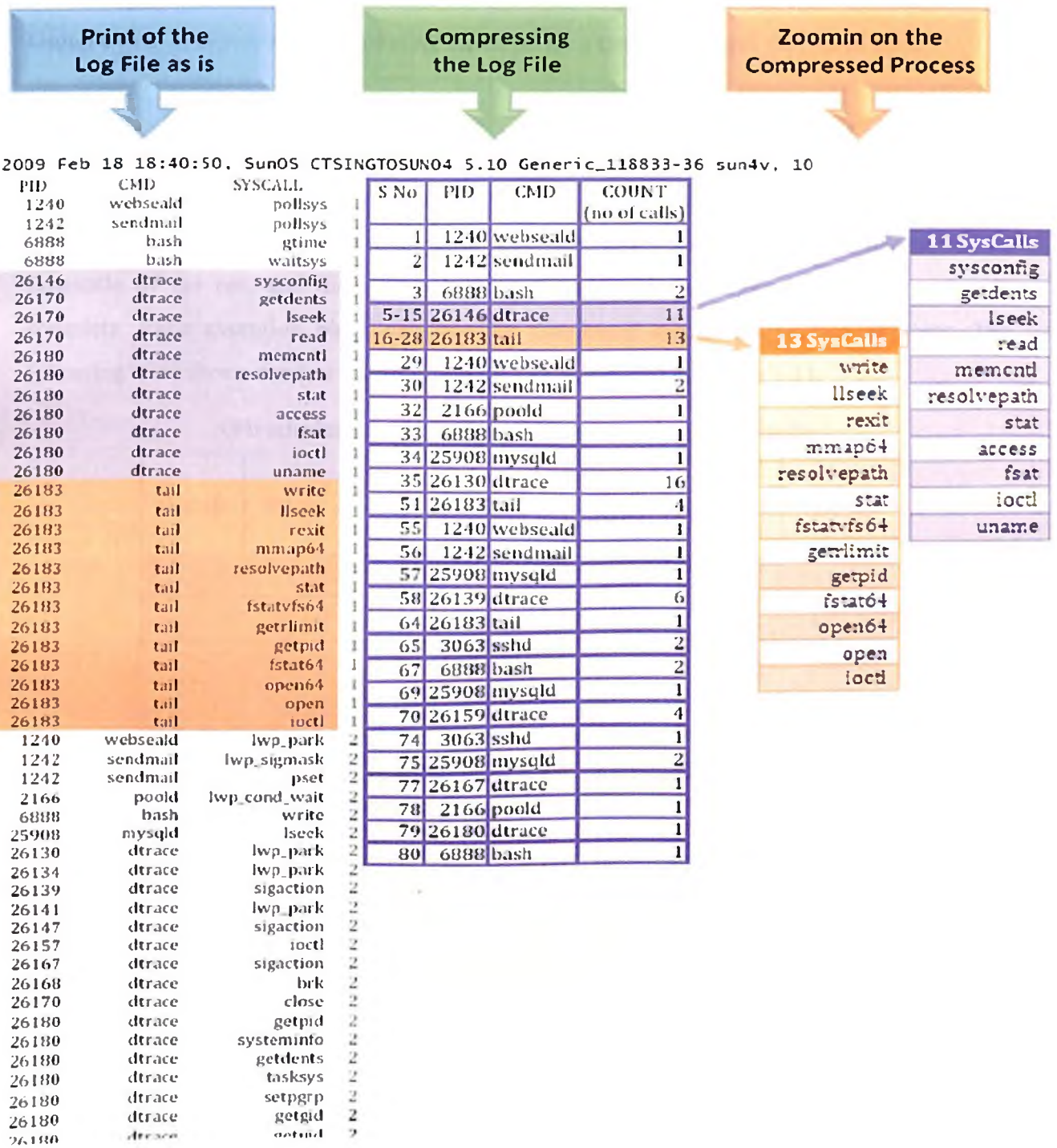
The following can be inferred from the records: (1) Adjacency of PIDs - The first command and the second command differ in PID by 2, indicating that webseald is followed by the operating system spawning the sendmail process. (2) Longevity of Run – The first two records are polling for a resource as seen in the SYSCALL command and have run for one cycle as indicated in COUNT.

A second look at from the figure below shows Adjacency of PIDs and Longevity of Run for DTrace (1) the PIDs for the 9 cores span between of 26170 and 26180 and (2) the commands run for a longevity of 11 (sum of count = 11) *i.e.* from record number 5 to record number 15.

RECORD NUMBER	PID	CMD	SYSCALL	COUNT
1	1240	webseald	pollsys	1
2	1242	sendmail	pollsys	1
3	6888	bash	gtime	1
4	6888	bash	waitsys	1
5	26146	dtrace	sysconfig	1
6	26170	dtrace	getdents	1
7	26170	dtrace	lseek	1
8	26170	dtrace	read	1
9	26180	dtrace	memcnt	1
10	26180	dtrace	resolvepath	1
11	26180	dtrace	stat	1
12	26180	dtrace	access	1
13	26180	dtrace	fsat	1
14	26180	dtrace	ioctl	1
15	26180	dtrace	uname	1

Filtering – Extraction and Grouping of desired processes

Expanding on the above, the following figures can be split into three sections. On the left, is a snapshot of the log file as is. These records contain the commands CMD for the 27 unique system calls (SYSCALL).



This single contiguous chain of commands (CMDs) can be compressed based on adjacency of records as shown in the Centre section in the figure above. The right hand side shows how the processes can be restored from the original records.. Two commands *viz.* dtrace and tail represent this grouping.

Thus, the following can be inferred. (1) The CMDs are presented in the order of execution. (2) Group based on adjacency and summing the count for a CMD compresses the record table.

Analysis – Recreating the Chronology

syscall_by_process.csv contains list of 37 unique commands (CMDs) within 91 unique system calls (SYSCALLs) run in 5209 (=112+503+622+1279+2693 = No of records) slots with 83747 (= Sum of COUNT). The SYSCALLs occur in chronological sequence. When SYSCALLs are superimposed on the timescale of the run, and distributed across processors, the chronological reconstruction is complete. Each execution block can now be distributed across the number of cores. The following table shows the data for TS1 through TS5 as detailed in Figure 5-11.

CPU schedules for 5 time intervals observed in DTrace probe

Run ID	Status	PID	CMD	No of CMDs	No of system calls	TIMING
TS1	Start 1	1240	webseald	112	3061	18:40:50
	Stop 1	26180	dtrace			18:41:00
TS2	Start 2	7	svc.startd	503	12418	18:41:21
	Stop 2	26212	dtrace			18:41:31
TS3	Start 3	126	nscd	622	8383	18:41:55
	Stop 3	26275	dtrace			18:42:05
TS4	Start 4	7	svc.startd	1279	17308	18:42:29
	Stop 4	26721	sshd			18:42:39
TS5	Start 5	7	svc.startd	2693	42577	18:43:11
	Stop 5	3203	telnet			18:43:21

37 Unique commands that run on the various cores

S No	COMMAND	S No	COMMAND	S No	COMMAND	S No	COMMAND
1	webseald	11	utmpd	21	syslogd	31	svc.configd
2	sendmail	12	fmd	22	dispqlen_by_cpu	32	newprocess_count
3	bash	13	inetd	23	interrupt_by_cpu	33	readb_by_process
4	dtrace	14	ls	24	sysinfo_by_process	34	syscall_errors
5	tail	15	login	25	sample_process	35	MySQL
6	poold	16	sh	26	vminfo_by_proces	36	t.sql
7	MySQLd	17	quota	27	writeb_by_process	37	minf_by_process
8	sshd	18	sed	28	cat		
9	svc.startd	19	expr	29	mail		
10	nscd	20	printf	30	telnet		

List of 87 System calls that are executed during observation by probes

List of system calls					
accept	access	acl	alarm	brk	c2audit
chdir	chown	close	connect	doorfs	dup
exece	fcntl	fork1	fsat	fstat	fstat64
fstatvfs64	getdents	getdents64	getgid	getmsg	getpid
getrlimit	getsockname	getuid	ioctl	llseek	lseek
lstat64	lwp_cond_signal	lwp_cond_wait	lwp_continue	lwp_create	lwp_exit
lwp_kill	lwp_park	lwp_self	lwp_sigmask	memcntl	mmap
mmap64	modctl	mprotect	munmap	nanosleep	open
open64	p_online	pathconf	pipe	pread	priocntlsys
privsys	pset	putmsg	read	recv	resolvepath
rexit	schedctl	send	setcontext	setegid	setgid
setgroups	setpgrp	setsockopt	setuid	shutdown	sigaction
sigaltstack	sigpending	sigwait	so_socket	stat	stat64
systeminfo	tasksys	times	umask	umount2	uname
unlink	write	zone			

Categorizing into desired processes

The next step is to group processes (CMDs) from syscall_by_process.csv based on three categories. (1) Those related to the MySQL like MySQL, MySQLd, t.sql and expr. (2) Those related to applications but NOT related to MySQL like poold, printf etc (3) Core Operating system processes like poold, telnetd etc. This grouping is done as observed from the application level.

When a sequence of records occurs for a group of MySQL related CMDs (like t.sql, MySQL and MySQLd) as the syscall_by_process. log in our test bed, the group can result in a cluster of MySQL related processes. The figure on a time-scale for "related MySQL calls in a cluster" depicts how the cluster can be identified. Two important statistical methods are used. The first – to get the number of clusters – is done by identifying set of sequence calls whose sequence count does not vary more a sequence count 200. Noise is further filtered on bidimensional data, based on the COMMAND type and the COMMAND sequence ID to obtain variance of the PID less than 100 for processes within a sequence.

The figure "Execution of only significant Cluster of CMDs", identifies the significant MySQL queries – whose system calls – sum of whose COUNT continuously – is greater than 100. This indicates how a CMD like MySQL sustains its execution in the core.

The number of schedules of MySQL is indicated by the number of green bars. This indicates how frequently MySQL based processes are scheduled and is an indicator of the effectiveness of scheduling MySQL processes which is parameterised as the Core Configuration Effectiveness (CCE) detailed in Section 5.3.1.

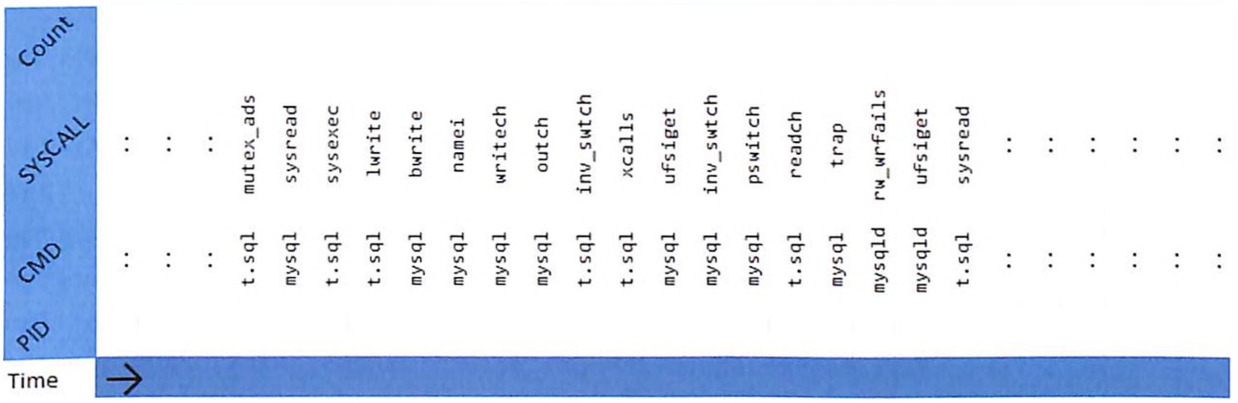
The efficiency of any of the MySQL related CMDs will determine the overall efficiency of that particular command in the Multi-core scenario (refer Intra Process Efficiency –Refer Section 5.3.2). For example, the 5193rd record of the syscall_by_process log indicates a COUNT of 11518.

The efficiency of 11518 executions depends on the Cache miss in these clock cycles. This efficiency is represented by Intra-Process Efficiency (IPE).

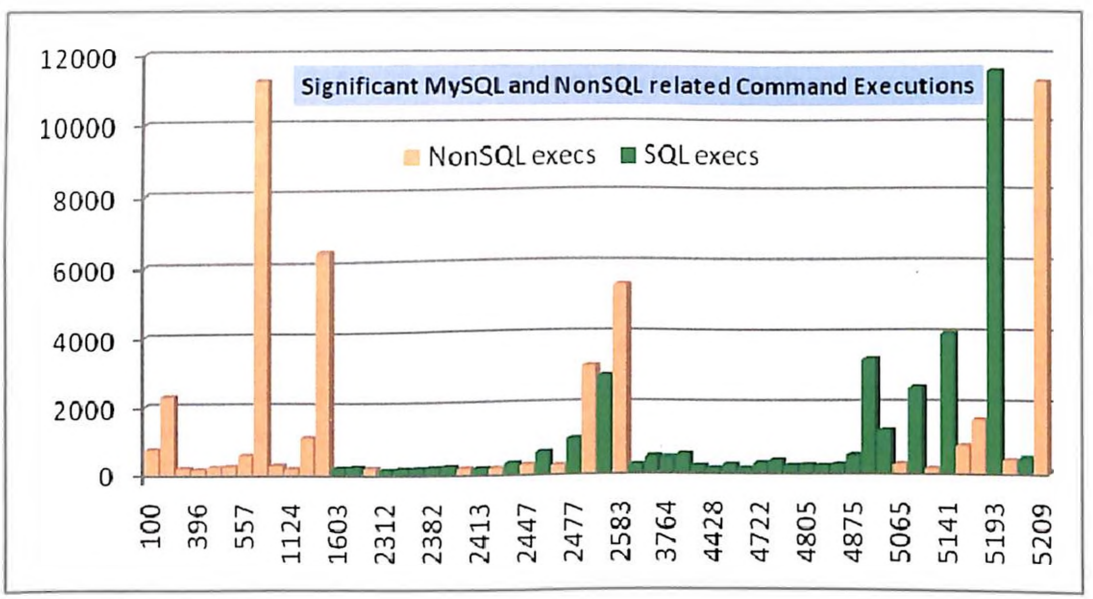
From the figure "Execution of only significant Cluster of CMDs", we can now determine the performance of the execution of MySQL on the Multi-core processor.

Command	bwrite	inv_swtch	lread	lwrite	mutex_ad	namei	nthreads	outch	pswitch	readch	rw_wrfails	sysexec	sysfork	sysread	syswrite	Trap	ufsdirblk	ufsiget	writech	xcalls
mysql																				
mysqld																				
t.sql																				

List of MySQL related system calls



Related MySQL calls in a cluster across a time scale.



Execution of only significant Cluster of CMDs (with system calls > 100)

APPENDIX VI

PRE-BUILT DTRACE PROBES FOR APACHE AND POSTGRESQL

There are about 154 pre built Apache Probes and following are the representative examples to show the probes related to connections, ports, and transactions.

- ap*::ap_run_child_init:child_init-entry()
- ap*::ap_run_child_init:child_init-return(int)
- ap*::ap_run_create_connection:create_connection-dispatch-complete(char *, int)
- ap*::ap_run_create_connection:create_connection-dispatch-invoke(char *)
- ap*::ap_run_create_connection:create_connection-entry()
- ap*::ap_run_create_connection:create_connection-return(int)
- ap*::ap_run_create_request:create_request-dispatch-complete(char *, int)
- ap*::ap_run_create_request:create_request-dispatch-invoke(char *)
- ap*::ap_run_create_request:create_request-entry()
- ap*::ap_run_create_request:create_request-return(int)
- ap*::ap_run_default_port:default_port-dispatch-complete(char *, int)
- ap*::ap_run_default_port:default_port-dispatch-invoke(char *)
- ap*::ap_run_default_port:default_port-entry()
- ap*::ap_run_default_port:default_port-return(int)
- ap*::ap_run_log_transaction:log_transaction-dispatch-complete(char *, int)
- ap*::ap_run_log_transaction:log_transaction-dispatch-invoke(char *)
- ap*::ap_run_log_transaction:log_transaction-entry()
- ap*::ap_run_log_transaction:log_transaction-return(int)
- ap*::ap_run_map_to_storage:map_to_storage-dispatch-complete(char *, int)
- ap*::ap_run_map_to_storage:map_to_storage-dispatch-invoke(char *)
- ap*::ap_run_map_to_storage:map_to_storage-entry()
- ap*::ap_run_map_to_storage:map_to_storage-return(int)
- ap*::ap_run_pre_connection:pre_connection-dispatch-complete(char *, int)
- ap*::ap_run_pre_connection:pre_connection-dispatch-invoke(char *)
- ap*::ap_run_pre_connection:pre_connection-entry()
- ap*::ap_run_pre_connection:pre_connection-return(int)
- ap*::ap_run_process_connection:process_connection-dispatch-complete(char *, int)
- ap*::ap_run_process_connection:process_connection-dispatch-invoke(char *)

- ap*::ap_run_process_connection:process_connection-entry()
- ap*::ap_run_process_connection:process_connection-return(int)

There are 49 prebuilt PostgreSQL DTrace probes, and some of themAREgiven below as a representation of them.

- postgresql*:::statement-start(const char *)
  - Fires any time SQL is executed on the server. copyinstr(arg0) is the query.
- postgresql*:::mark-dirty(uint32_t)
  - Fires when a buffer in the shared buffer pool is marked dirty for the first time. The first argument is the buffer (id).
- postgresql*:::local-mark-dirty(uint32_t)
  - When a local buffer in the shared buffer pool is marked dirty for the first time. The first argument is the buffer (id).
- postgresql*:::slru-readpage-entry(uintptr_t, uint32_t, uint32_t, uint32_t)
  - Fires on the entry to the slru SimpleLruReadPage function. arg0 is a pointer to the PostgreSQL SlruCtl. arg1 is the page number, arg2 is 0 or 1, indicating if the page needs to be writable. arg3 is the transaction id (xid).
- postgresql*:::slru-readpage-return(uint32_t)
  - Fires when the slru SimpleLruReadPage function returns. arg0 is the slot number of the page returned.
- postgresql*:::slru-readpage-ro(uintptr_t, uint32_t, uint32_t)
  - Fires when the slru SimpleLruReadPage_ReadOnly function is entered. arg0 is a pointer to the PostgreSQL SlruCtl. arg1 is the page number. arg2 is the transaction id (xid).
- postgresql*:::slru-writepage-entry(uintptr_t, uint32_t, uint32_t)
  - Fires when the slru SimpleLruWritePage function is entered. arg0 is a pointer to the PostgreSQL SlruCtl. arg1 is the page number. arg2 is the slot number.
- postgresql*:::slru-writepage-return()
  - Fires when the slru SimpleLruWritePage function returns.
- postgresql*:::slru-readpage-physical-entry(uintptr_t, char *, uint32_t, uint32_t)
  - Fires when the slru SlruPhysicalReadPage function is entered. arg0 is a pointer to the PostgreSQL SlruCtl. copyinstr(arg1) is the pathname of the file being read. arg2 is the page number. arg3 is the slot number.
- postgresql*:::slru-readpage-physical-return(uint32_t, uint32_t, uint32_t)
  - Fires when the slru SlruPhysicalReadPage function returns. arg0 is 0 or 1 indicating success. arg1 is the internal error cause (only valid if arg0 is 0). arg2 is the system errno (only valid if arg0 is 0).

APPENDIX VII

LIST OF SCRIPTS

Threads.py

```
#The script (threads.py) send request to server xmlrpc server, it will span
n number of #threads and send request to the xmlrpc server.
#!/usr/bin/python
import threading
import random
import sys
import xmlrpclib
import time
import os, sys

diff = 1
requests = 100
iterations = 1

rpc_srv = xmlrpclib.ServerProxy("http://127.0.0.1:7080")

class MyThread(threading.Thread):
   def __init__(self, initial_value):
      threading.Thread.__init__(self)
      self.num = initial_value

   def run(self):
      '''
      num = random.randrange(0,999)
      if num < 100 and num >= 10: num = '0'+str(num)
      elif num < 10: num = '0'+str(num)+'0'
      else : num = str(num)
      '''
      print self.num
      print str(rpc_srv.get_count(self.num)[0][0]) + ' rows found for card
number ' + str(self.num)
```

```
start_time = time.ctime()
for x in range(iterations):
      threadList = []
      num_threads=open('threads.txt').read()
      #Comment the below line for reading requests from threads.txt
      #Uncomment the below line for reading request from request variable
mentioned above.
      #num_threads = str(requests)
      fp = open('time.txt', 'a')
      if int(num_threads):
         print 'running ' + num_threads + ' requests...'
         fp.write('run #' + str(x) + ':\n')
         fp.write('running ' + num_threads + ' requests...\n')
         fp.write('start time:' + time.ctime() + '\n')
         time1 = time.time()
         x=x+1
         for j in xrange(int(num_threads)):
            if x==9:
               mt=MyThread(x*100 + 49 + j)
            else:
              mt=MyThread((x+1)*100 + j)
            threadList.append(mt)
            mt.start()

         for each_thread in threadList:
            each_thread.join()
         fp.write('end time:' + time.ctime() + '\n\n')
         time2 = time.time()

         delta = int(time2 - time1)

         if delta % diff != 0:
            time.sleep(delta % diff)

      time.sleep(diff)
      a = rpc_srv.get_threads()
      b = rpc_srv.del_threads()
      fp.close()

      if a:
         print a[0][0]
         open('threads.txt', 'w').write(str(a[0][0]))
```

Python - DTracing Python

These scripts DTrace the Python programming language, and require a version of Python which has been built with DTrace probes.

The Python DTrace provider was originally written by John Levon, and was integrated into Solaris Nevada in build 65. If you are on a different OS with DTrace and would like to use these scripts, you could download Python and the Python DTrace provider patch listed in the comments here,

```
http://blogs.sun.com/levon/entry/python_and_dtrace_in_build
```

You will need patch and build Python for these probes to work.
Or, check if a pre-built package is available someone on opensolaris.org.

Since the DTrace Python provider may be developed further, there is a chance that it has changed slightly by the time you are reading this, causing these scripts to either break or behave oddly. Firstly, check for newer versions of the DTraceToolkit; if it hasn't been updated and you need to use these scripts immediately, then updating them shouldn't take too long. The following was the state of the provider when these scripts were written - check for changes and update the scripts accordingly,

```
provider python {
    probe function-entry(file, subroutine, lineno)
    probe function-return(file, subroutine, lineno)
};
```

xmlrpc_server

```
#The script (xmlrpc_server.py) will start the server on the windows machine
which connects the MySQL db on #solaris server and listening client request
and send back the result to client.
#
from twisted.web import xmlrpc, server
from twisted.internet import reactor
from twisted.enterprise import adbapi
dbpool = adbapi.ConnectionPool("MySQLdb", cp_max=500, cp_reconnect=True,
db="my_cc", host='10.237.214.157', user='root')
dbpool2 = adbapi.ConnectionPool("MySQLdb", cp_max=2, cp_reconnect=True,
db="my_cc2", host='10.237.214.157', user='root')

class Example(xmlrpc.XMLRPC):
    """An example object to be published."""
    def xmlrpc_echo(self, x):
        """Return all passed args."""
        return x
    def xmlrpc_block(self, duration=10):
        """block the instance for a specified duration"""
        import time
        time.sleep(duration)
        return "i slept %s seconds!" % (str(duration))

    def getCount(self, num):
        return dbpool.runQuery("select count(distinct(ch_cardno)) from
ch_transaction where ch_cardno like '%" + str(num) + "%'")
    def getThreads(self):
        return dbpool2.runQuery("select threads from tmp_table")
    def deleteThreads(self):
        return dbpool2.runQuery("delete from tmp_table")
    def xmlrpc_get_count(self, x):
        return self.getCount(x).addCallback(lambda x: x)
    def xmlrpc_get_threads(self):
        return self.getThreads().addCallback(lambda x: x)
    def xmlrpc_del_threads(self):
        return self.deleteThreads().addCallback(lambda x: x)

# this only runs if the module was *not* imported
if __name__ == '__main__':
    r = Example()
    reactor.listenTCP(7080, server.Site(r))
    reactor.run()
```

APPENDIX VIII

MySQL Deployment configuration parameters

Following is the list of twenty three deployment configuration parameters available for managing the performance of MySQL database server.

1. character_set_client
2. iinnodb_additional_mem_pool_size
3. innodb_buffer_pool_size
4. innodb_file_io_threads
5. innodb_log_buffer_size
6. innodb_thread_concurrency
7. innodb_thread_sleep_delay
8. join_buffer_size
9. key_cache_block_size
10. key_cache_division_limit
11. max_binlog_cache_size
12. query_cache_size
13. query_cache_type
14. query_cache_wlock_invalidate
15. query_prealloc_size
16. read_buffer_size
17. shared_memory
18. sort_buffer_size
19. thread_cache_size
20. thread_concurrency
21. bdb_cache_size
22. binlog_cache_size
23. bulk_insert_buffer_size

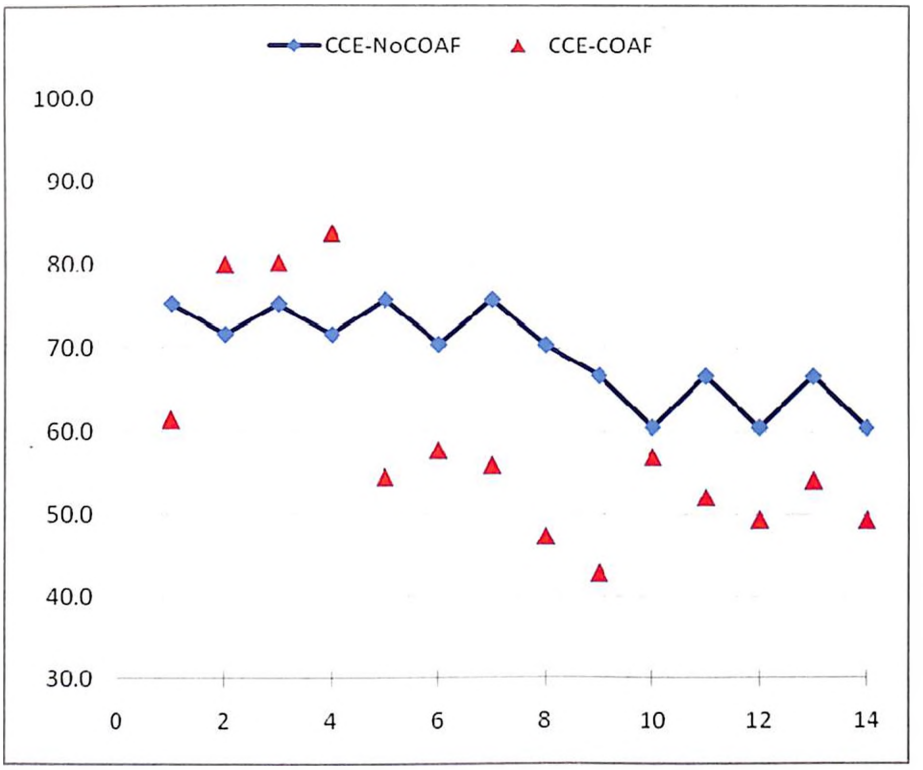
APPENDIX IX

STATISTICAL ANALYSIS FOR VERIFICATION / REJECTION OF THE NULL HYPOTHESIS.

We changed the deployment parameters viz. ThreadPool Size to verify if the *performance* of *web service based enterprise applications* from the application layer varies when the deployment parameter is varied. Our null hypothesis predicts that the variation in the efficiency should be no greater than the 5% of the efficiency for a sample size of 13 experiments.

Cores	CCE-NC	CCE-C	IPE-NC	IPE-C
16	75.3	61.5	79.3	89.8
16	71.6	80.0	66.4	90.6
16	75.3	80.2	79.3	92.0
16	71.6	83.8	66.4	85.0
17	75.8	54.6	75.6	89.3
17	70.4	57.8	60.5	88.5
17	75.8	56.0	75.6	89.1
17	70.4	47.4	60.5	93.0
32	66.7	43.0	71.9	92.8
32	60.6	57.1	57.2	50.5
32	66.7	52.2	71.9	34.3
32	60.6	49.5	57.2	93.4
32	66.7	54.3	71.9	94.2
32	60.6	49.5	57.2	34.3

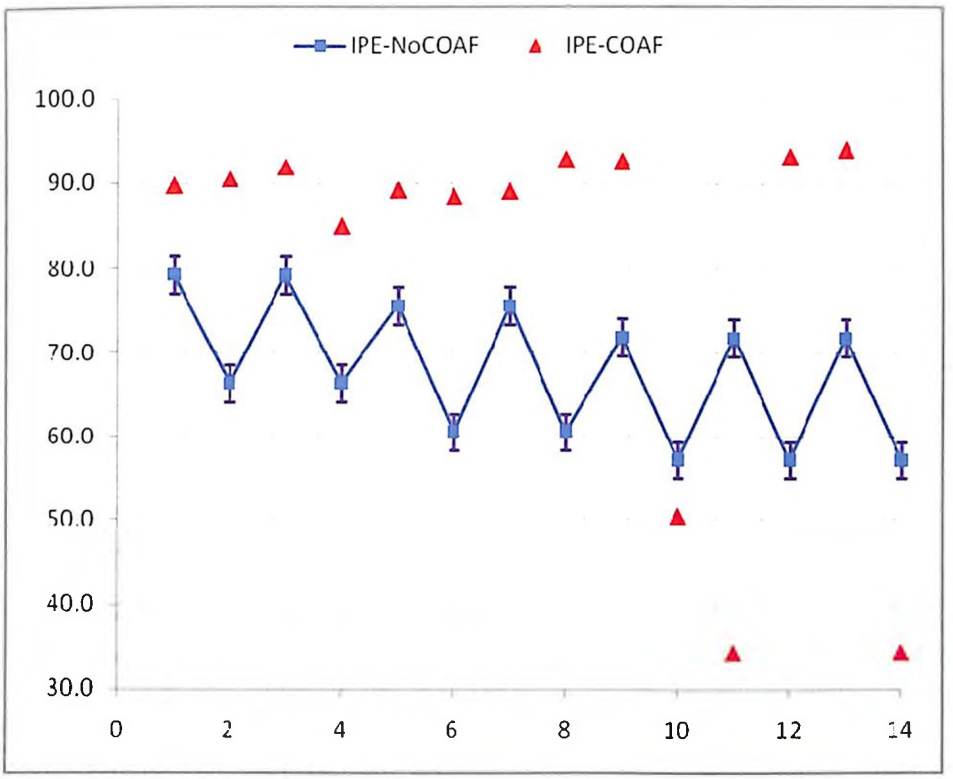
The hypothesised value for the *Core Configuration Effectiveness* with the *std err* is shown in the following chart, wherein all the values of the NoCOAF are shown with the stderr, whereas the values of the experiment with COAF are shown as actual. It can be observed from the experiments that by only changing deployment configurations at the application level, and leverage the power of Multi-core infrastructure, the variation in CCE is significant.



t-Test: Two-Sample Assuming Equal Variances for CCE		
	Without COAF	*With COAF*
Mean	69.15	59.06428571
Variance	32.04115385	168.0363187
Observations	14	14
Pooled Variance	100.0387363	
Hypothesized Mean Difference	3	
df	26	
t Stat	1.874340796	
P(T<=t) one-tail	0.036080953	
t Critical one-tail	1.705617901	
P(T<=t) two-tail	0.072161906	
t Critical two-tail	2.055529418	

Similarly, when the infrastructure parameters at the application level, viz. the max threads is configured automatically, the hypothesised value for the *IntraProcessEfficiency and std err* is shown for NoCOAF in the following chart. The values obtained from the experiment with COAF are shown without the stderr. It can be observed from these experiments that by only changing deployment thread configurations, at the application level, and the power of Multi-core

infrastructure is leverage in the form of reduced Cache misses, i.e. the variation in IPE is significant as shown in the table that follows.



t-Test: Two-Sample Assuming Equal Variances for IPE		
	Without COAF	*With COAF*
Mean	69.15	59.06428571
Variance	32.04115385	168.0363187
Observations	14	14
Pooled Variance	100.0387363	
Hypothesized Mean Difference	3	
df	26	
t Stat	1.874340796	
P(T<=t) one-tail	0.036080953	
t Critical one-tail	1.705617901	
P(T<=t) two-tail	0.072161906	
t Critical two-tail	2.055529418	

LIST OF PUBLICATIONS / PRESENTATIONS

1. Venugopal, S., and Ganesan, K. Tools for observing kernel behaviour. In *Proceedings of National Conference on Recent Trends in Information Technology (NCRTIT '07)* (August 2007), Chennai, India.
2. Venugopal, S., and Desikan, S.K. Modernization challenges in transition to the HD world. *Lecture notes in Broadcast Engineering Society*, Mar 2010.
3. Venugopal, S., Desikan, S.K. and Ganesan, K. Connection Oriented Framework for Effectively Using Multicore in the Enterprise. In *Proceedings of 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS '11)* (February 2011), IEEE. Paris, DOI:10.1109/NTMS.2011.5720651.
4. Venugopal, S., Desikan, S.K. and Ganesan, K. Effective Migration of Enterprise Applications in Multicore Cloud. In *Proceedings of International Workshop on Cloud Computing & Future of work (UCC '11)* (December 2011), IEEE, Melbourne, 463-468, ISBN: 978-0-7695-4592-9.
5. Venugopal, S., Desikan, S.K. and Ganesan, K. Connection Oriented Framework for enterprise applications. *International Journal of Advanced Computing. Volume 4, Issue 2, Dec 2012* Accepted for publishing.

BRIEF BIOGRAPHY OF THE STUDENT

Name: P. V. Suresh (Suresh Venugopal)

P. V. Suresh received AMIE from Institute of Engineers, India in 1988, and ME from PSG College of Technology, Coimbatore, India in 1992. He has more than twenty years of professional experience in industry. He is currently Vice President and Chief Architect for Technology practice at Cognizant Technology Solutions, India. In his current role, he oversees the technology, architecture and delivery of the projects to online, customers like Amazon, eBay, IAC, Google, Paypal etc. and hitech customers like Agilent, Logitech, Tektronix, etc. of Cognizant. Additionally, he is the mentor for Global Technology Office of Cognizant, which drives the technology and research initiatives of Cognizant worldwide. Prior to joining Cognizant in 2003, he held various technology and research positions at Planetasia, Parametric Technology Corporation, Electronica and Caterpiller Corporation. As General Manager of Engineering at Planetasia he was responsible for all technology and architecture decisions for the implementations done by Planetasia. At Parametric Technology Corporation, Boston, USA, he was release manager for Wildfire line of product lines based Boston, USA. At Electronica he designed and developed the PC based control system for CNC machines, which was the pioneering research effort in India during that time.

His research interests are distributed computing, automation, performance modelling of web based applications, and massive computing architectures. He has conducted multiple workshops as part of his profession to various research and development departments of Cognizant customers. He evangelises on agile methodologies and usage of open source, across student and developer community, and speaks in Industry forums like SPIN conferences, Linux forums, etc. He is the member of Institution of Engineers, and International Association of System Architects.

BRIEF BIOGRAPHY OF THE SUPERVISOR

Name: Dr. G. Karthikeyan

Dr. G. Karthikeyan received the BS from Anna University in 1982, MS and PhD from Indian Institute of Science, Bangalore, India in 1984 and 1989, respectively, and a post doctoral fellowship from the Ohio State University. He has more than twenty years of professional experience in industry and teaching. He is currently a Program Manager in Cognizant Technology Solutions at Boston, USA. In his current role, he oversees the delivery of the projects to Cognizant customers within budget, schedule and cost. Additionally, he consults on developing and implementing optimization and scheduling algorithms for specific problems of Cognizant customers. Prior to joining Cognizant in 2000, he was a Director in Technology Gateway and an Assistant Professor in Industrial Engineering at the College of Engineering, Guindy for more than ten years. He was responsible for teaching to Industrial engineering students in post graduate and graduate students. During his tenure at Anna University for more than ten years, he was closely involved the implementation of largest IT project for Anna University at the point in time, sponsored by Govt of India to automate the technology collaboration mechanisms across Universities and colleges.

His research interests are optimization modelling, performance engineering, customer requirements analysis, demand forecasting, reliability, and highly available Web service applications. He has more than 45 publications in leading National and International Journals and Conferences, and has a patent on product quality assurance. He was awarded the UGC Research Fellowship in Engineering & Technology, the BOYCAST Fellowship by the DST, and the State Scientist Award in 1998. Dr. G. Karthikeyan is a Fellow of the Institution of Engineers, and Indian Institute of Materials Management. He has supervised more than 50 students for the research projects including 3 PhD students.