

IMPLEMENTATION OF REAL-TIME CLUSTER MANAGEMENT SYSTEM WITH CRASH RECOVERY

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*

of

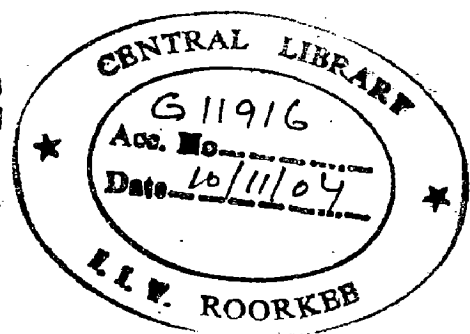
MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

GLADVIN C DURAI



**IIT Roorkee - CDAC, NOIDA,
c-56/1, "Anusandhan Bhawan"
Sector 62, Noida-201 307**

JUNE, 2004

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this dissertation titled "**Implementation of Real Time Cluster Management System with Crash Recovery**", in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Information Technology**, submitted in **IIT, Roorkee – CDAC, Noida**, is an authentic record of my own work carried out during the period from May 2003 to May 2004 under the guidance of **Maj. Gen. K. N. Singh** Chairman & MD, Next Gen Media Alliances Pvt. Ltd., New Delhi.

The matter embodied in this dissertation has not been submitted by me for award of any other degree or diploma.

Date: 26 - JUNE - 2004

Place: Noida

G. Gladwin
(Gladwin C Durai)

CERTIFICATE

This is to certify that the above statement made by candidate is correct to the best of my knowledge and belief.

for

R. Agarwal

Date:

Place: New Delhi

Supervisor:

(Maj. Gen. K. N. Singh)
Chairman & MD
NextGen Media Alliances Pvt. Ltd.

V. N. Shukla

Co-Supervisor:

(Mr. V. N. Shukla)
Director (Spl. Appl.)
CDAC, Noida

ACKNOWLEDGEMENTS

I hereby take the privilege to express my deepest sense of gratitude to **Prof. PREM VRAT**, Director, Indian Institute of Technology, Roorkee, and **MR. R.K.VERMA**, Executive Director, CDAC, Noida for providing me with this valuable opportunity to carry out this work. I am very grateful to **Prof. A.K.AWASTI**, Programme Director, **Prof. R.P. AGARWAL**, course coordinator, IIT, Roorkee and **MR. V.N.SHUKLA**, course coordinator, CDAC, NOIDA for providing the best of the facilities for the completion of this work and constant encouragement towards the goal.

I express my sincere thanks and gratitude to **Mr. RAGHAVENDRA AGARWALA**, Chief Technical Officer and also to my Project Guide **Maj. Gen. K.N.SINGH**, Chairman, Next Gen Media Alliances Pvt. Ltd., New Delhi. I am thankful to **Mr. MUNISH KUMAR**, **Dr. POONAM RANI GUPTA**, Associate Professor, CDAC, Noida. They gave inspiration and guidance throughout the progress of this project. I feel very much privileged to associate myself for completing this project successfully.

I thank my co-analysts, programmers and other team members who were with me while configuration, trouble shooting, and in integration of the modules.

I owe special thanks to my friends, all of my classmates and other friends who have helped me formulate my ideas and have been a constant support. Thanks to my family members for their moral support. Last but not the least; I thank almighty for being on my side from the conception of this project to this implementation.

(GLADVIN C DURAI)

Enrolment No. 029005

Contents

Candidate's Declaration	i
Acknowledgements	ii
Abstract	v
1. Introduction	1
1.1 Cluster Management System	1
1.2 Present world status and Problems	2
1.3 How the dissertation solves the above problems	2
1.4 Dissertation Objective	3
1.5 Report Organization	3
2. Literature Survey	5
2.1 Today's Scenario	5
2.1.1 Evolution of Cluster Computing	5
2.1.2 Common Cluster Issues and Common Cluster Solutions	7
2.2 Native Solutions	7
2.3 Application Development Platforms	10
2.4 Application Servers	11
2.4.1 BEA Architecture	12
2.4.2 Pramati Server	15
2.4.3 On Demand Distributed Computing with WebSphere	16
2.4.4 Turboworx	24
3. Analysis	25
3.1 High End Computing	25
3.2 Scheduling Algorithms	27
3.3 Characteristic of a node	29
3.4 Load sharing issue	29
3.5 System Scalability	31
3.6 Flexibility vs. Performance in communication Protocol	31
3.7 Peer-to-Peer computing	32

4. Design	33
4.1 Interaction System	33
4.1.1 Messages	33
4.1.2 Interoperability with XML	34
4.1.3 Inter system peer-to-peer communication	34
4.2 Load Management	36
4.2.1 Load Balancing and Informing	36
4.2.2 Load Restriction	37
4.3 Naming Conventions	37
4.4 Crash Recovery	37
4.5 Transparency to application developers	38
4.6 Administration	39
4.6.1 Local administration and remote administration UI	39
4.6.2 Self-managing, Self-organizing, Self-healing	39
4.7 Design of Modules	40
4.7.1 Data Flow	40
4.7.2 Session Manager	41
4.7.3 Generic Socket	43
4.7.4 Process Scheduler	44
4.8 Summary	46
5. Implementation	47
5.1 Application Programmers guide	47
5.2 Administrator's guide	47
5.3 Remote Administrator communication	48
5.4 Marshalling and de-marshalling of data to transmit	48
6. Results and discussions	49
6.1 Starting application manager	49
6.2 Starting programs in computer	49
6.3 Flow of request.	50
7. Conclusions	51
References	

Abstract

The present day organizations and large e-business requires the computing power of servers to increase and increase. But can the service providers keep replacing with new advanced system. If they are having Real Time Cluster Management System then they can upgrade through adding-up instead of replacing the existing servers.

The Internet had in late 60's private networks, which were not interoperable. And then static web pages were published on the Internet through interoperable technologies. And then portals came and recently e-businesses. The future is going towards sharing of computing resources like SETI@home (Search for Extra-Terrestrial Intelligence) through distributing the computational power.

Suppose a Web Servers like a mail server or a search engine has the server applications run on a single computer. Now as the processing load increases there is a need to upgrade the system, thus the existing system has to be replaced, which is costly and wastage of resources. In this case you have limitation to scale the processing resource.

All the server applications are made exclusively to work on a single host processing units. Through the Cluster Management System, many instances of same code is run on different systems and thus provide scalability of processing power. This enhances the scalability and availability of mission-critical, TCP/IP-based services.

The issues needed to be taken care in this mode are very unique like, interoperability at Operating System level, Programming Language level; asynchronous messaging between hosts; abstraction layer to application and to clients; session maintenance between cluster applications and process scheduling and various other issues like availability monitoring; automatic redistribution of client traffic to the surviving hosts, remote controllability and maintaining logs of events.

This report explains one method by which Real Time Cluster Management System is achieved and how it is better with respect to other leading techniques.

INTRODUCTION

1.1 Cluster Management System

The real-time cluster management system can be defined as the combination of network and software techniques to leverage the power of a cluster of nodes. Nodes work together along with load balancer and act as a single virtual server. As more and more processing units are added to the cluster they instead of substituting the processing power, they add up to the processing power of the cluster. Internet server programs supporting mission-critical applications such as financial transactions, database access, corporate intranets, and other key functions must run 24 hours a day, seven days a week. Clustering enables a group of independent servers to be managed as a single system. Also for those institutes who are doing researches, defence organisations and for those who are doing batch processing over night require a heavy amount of processing resource, so it can be bought from a commercial site. Figure 1.1 shows a typical Cluster.

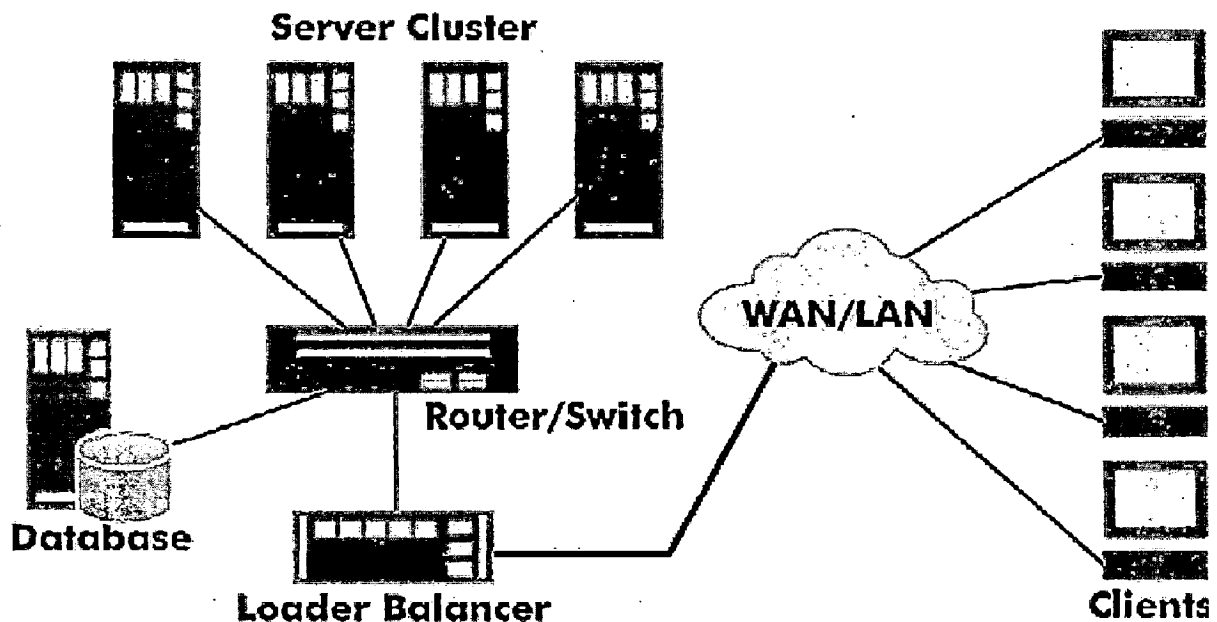


Figure 1.1: Single load balancer takes request and routes to the free host.

Constantly growing organisations use this technology when they want to provide cost efficiency, scalability and availability of mission-critical, TCP/IP-based services.

1.2 Present world status and Problems

The motivation of the dissertation is that, the existing solutions have drawbacks and those drawbacks are removed from this dissertation. The solutions like Round Robin DNS [URL6], which is a centralised architecture, redirects only once and the transactions have to be maintained by the applications themselves. This is a simple and cheapest solution. This is not a robust solution and has no recovery mechanism. There is no asynchronous messaging which is an integral part of Cluster Management Systems. The Network Load Balancing package [URL7, URL8, URL15] embedded in Windows 2000 Advanced Server and in Data Server has the advantage to have multiple concurrent systems waiting on the network for all the receive and except the one system everyone else rejects the request. This is adaptive and makes one level redirection. NLB provides failover support for MS-SQL database. The problems in this are interoperability with non-Microsoft system and the scalability is limited by ports hence supports maximum of 32 systems. MOSIX [URL9] is another research project done in a university at Israel. This supports process level redirection and multiple level redirections too and at present demonstrated with 72 processors. It is working in UNIX, BSD, Linux machines. This is still in development stage. This has no failover recovery. It is a fully distributed architecture. Other solutions called application servers do exist and they cost in the range of seven lakhs per host (Pramati web server [URL2]) to 18 lakhs per host (BEA web logic-server [URL3]), but they are not customisable to the larger extend.

1.3 How the dissertation solves the above problems

The interoperability is provided by passing XML [URL1] file transfer between systems and using byte streams. The parser used has the capability of parsing and de-parsing between objects into XML files. Hence this system could be used with the programming language objects of visual C++, Java, and Cocoa. Having buffers at the two ends does asynchronous messaging. Remote admin is provided to monitor the status of the systems. Shell prompt also provide input channel. Using shared databases provides the persistence of data. The load computation is application vice hence the load is more

exact. This accommodates systems with various configurations. Applications are provided with wrapper classes as abstraction while communication with data objects, hence application programmers could easily integrate with cluster system.

1.4 Dissertation Objective

The purpose of this study is to investigate and implement a less featured, more reliable and efficient way of achieving cluster management system and to overcome the limitations of conventional cluster management systems. Further to provide a convenient reporting and controlling mechanism across the cluster nodes.

1.5 Report Organization

In chapter 2 definition of cluster management system is given with regard to Real Time Clusters. It also discusses different design issues to be considered and a survey of existing cluster management systems. Chapter 3 covers the details of necessary concepts that help in working of the Cluster systems and explains the infrastructure required to implement Real Time Clusters. It also describes the peer-to-peer protocol to use and the various scheduling algorithms in existence. Chapter 4 discusses how the identification is done through naming conventions and the load management is designed. The interaction system's capabilities are specified one by one. It also explains the module vice interaction and association within the module of various components. Chapter 5 explains how a application programmer initiates the cluster system and how to send and receive data through the cluster system and the method by which different features are implemented in this work. Chapter 6 of the report includes the GUIs built and results obtained. Chapter 7 concludes the report mentioning how Real Time Cluster Management as implement through this approach has overcome the limitations mentioned in chapter 1 and gives further areas of improvement.

LITERATURE SURVEY

In this chapter a study of the requirement for Real Time Cluster Management System and various existing solutions is presented.

2.1 Today's Scenario

Most technical problems in engineering, science, medicine, and financial services are solved using computational workflows that integrate numerous related, but distinct application components. Modern computing environments are potentially excellent platforms for processing such workflows because they allow the work to be distributed among large numbers of highly capable independent machines. However, the tasks of integrating applications, building workflows, scheduling machines, moving data, and managing the entire distributed computing environment are daunting.

Parallel computing uses multiple computers or internal processors to solve problems at a greater computational speed than using a single computer or processor.

2.1.1 Evolution of Cluster Computing

In the past, organizations performed computing tasks in highly integrated enterprise computing centres. Although sophisticated distributed systems existed, such as command-and-control and reservation systems, and the Internet Domain Name System, these were specialized, niche entities. The Internet's rise and the emergence of e-business have, however, led to a growing awareness that an enterprise's IT infrastructure also encompasses external networks, resources, and services.

Initially, developers treated this new source of complexity as a network-centric phenomenon and attempted to construct intelligent networks that intersected with traditional enterprise IT data centres only at edge servers (the virtual private network server that connects an enterprise network to service provider resources), for example. These developers worked from the assumption that these servers could thus manage and

circumscribe the impact of e-business and the Internet on an enterprise's core IT infrastructure.

These attempts have generally failed because IT services decomposition is also occurring inside enterprise IT facilities. New applications are being developed for programming models, such as the Enterprise JavaBeans component model, that insulate the application from the underlying computing platform and support portable deployment across multiple platforms. Thus, for example, Web serving and caching applications target commodity servers rather than traditional mainframe computing platforms. Meanwhile, Web access to enterprise resources requires ever-faster request servicing, further driving the need to distribute and cache content closer to the network's edge.

The overall result is decomposition of a highly integrated internal IT infrastructure into a collection of heterogeneous and fragmented systems, often operated by different business units. Enterprises must then reintegrate these distributed servers and data resources with QoS, addressing issues of navigation, distributed security, and content distribution inside the enterprise as well as on external networks.

In parallel with these developments, enterprises require an increasingly robust IT infrastructure to handle the unpredictability and rapid growth associated with e-business ventures. Businesses are also expanding the scope and scale of their enterprise resource planning projects as they try to achieve better integration with customer-relationship-management, integrated-supply-chain, and existing core systems.

These developments have the aggregate effect of making the QoS traditionally associated with mainframe host-centric computing essential to the effective conduct of e-business across distributed computing resources, both inside and outside the enterprise. For example, enterprises must provide consistent response times to customers, despite workloads with significant deviations between average and peak utilization. Thus, they require flexible resource allocation in accordance with workload demands and priorities. Yet the current paradigm for delivering QoS to applications via the vertical integration of platform-specific components and services does not work in today's distributed environment: The decomposition of monolithic IT infrastructures is inconsistent with the delivery of QoS through vertical integration of services on a given platform.

Modern Pay Per Use model [10]

Companies such as Entropia hope to capitalize on distributed-computing technology by paying ordinary Web users for use of their spare computer processing cycles. The companies then sell access to the resulting Internet-based grid to commercial concerns such as genetics researchers.

2.1.2 Common Cluster Issues and Common Cluster Management Solutions

High-volume Web sites often use cluster of servers to support their architectures. A load balancer in front of such clusters directs requests to the various servers in a way that equalizes, as much as possible, the load placed on each.

There are two basic approaches to scaling Web clusters: adding more servers of the same type (scaling out, or horizontally) or upgrading the capacity of the servers in the cluster (scaling up, or vertically) [9].

Typical questions about Web cluster design include whether to use a large number of low-capacity servers or a small number of high-capacity costly ones to provide a given performance level? How many servers of a given type are required to provide a certain performance level at a given cost?

Common Cluster Management solutions are Native solutions like Network Load Balancing of Windows. Application development platforms like J2EE and Application Server like Weblogic Server.

2.2 Native Solutions

These are the ready-made solutions available for us. These architectures are less flexible and they have limitation in capabilities.

2.2.1 Network Load Balancing Architecture in Windows [URL8]

The clients are differentiated by request type and client related factors. Using this information "load sharing" is done by allocating some set of users for every node. Heart beat messages are passed among the servers to check each other node's availability. If the server node is offline then the clients retry the connections. Remote administration module is present to know the status and to configure the node. Some session has to be maintained by the applications themselves.

2.2.2 MOSIX [URL9]

Based on Shared memory multi-computer

Every process has Unique Home Node, where shared memory access requests are sent back. The transactions is kept at the common data store where each client depended data could be taken from.

Based on Distributed architecture

This is fully distributed architecture where every system is capable of making its own decision for redirection of request. And it is capable of communicating with the next system and naming conversion can independently make.

Implementation

Servers and workstations are used as a single cluster by installing the same "mosix.map" in all the computers, with the IP addresses of all the servers and all the workstations. Advantage/disadvantage: your workstation is part of the pool.

Servers are shared while workstations join or leave the cluster, e.g. from 5PM to 8AM. Use a simple script to decide whether MOSIX should be activated or deactivated in your workstation. Advantage/disadvantage: remote processes can use your workstation when you are not using it.

2.2.3 Other Solutions

2.2.3.1 PVM and MPI

PVM (Parallel Virtual Machine) and MPI (Message Passing Interface) are both popular and freely available parallel software development aids. The Oak Ridge National Laboratory developed PVM for internal use and released it to the public after some refinement. A committee known as the MPI Forum developed MPI as a standard. MPI has become more popular to use than PVM, owing mostly to its improved portability. Vendors develop proprietary versions of PVM that are optimized for their platforms, but systems developed using one vendor's PVM might not compile on another's platform. Vendor support of MPI, on the other hand, was required to meet strict interface standards.

2.2.3.2 THE OPENMP

A true multiprocessor offers multiple CPUs, each with equal access to a shared physical-memory area. Twenty years ago, Kai Li, a Yale doctoral student, proposed a shared-memory model for a multi-computer, which used custom software on a network of workstations. The software provided a layer of support for shared virtual memory that spanned his network of uniprocessors. Programs written for a multiprocessor would run more slowly on his system, but his system was a small fraction of a true MP's cost. His dissertation and subsequent publications influenced parallel architecture research for over a decade. OpenMP is a programming language based on this kind of model.

One key difference between MPI and OpenMP is the approach to exploiting parallelism in an application. MPI requires the developer to convert the entire application immediately. OpenMP allows an incremental conversion; the developer can convert, profile, and tune a large application in a stepwise fashion, simplifying the debugging and development process. If you decide to use OpenMP to develop a parallel system, you must either have access to a multiprocessor or run a custom software support layer that emulates these properties on a multi-computer.

2.2.4 Comparative study of existing Native solutions

Issues	RRDNS [URL7]	NLB [URL8]	MOSIX [URL9]
Architecture	Centralized	Fully distributed	Fully distributed
Load Distribution	Fixed	Adaptive	Adaptive
Valid Request protocols	http like disconnected protocols	TCP/IP	Process level
Examples	Popular proxy servers	Win 2000 advanced server	Data center server

Table 2.1: Comparison of Native Solutions

Fault Tolerant	Single point of failure	Redundancy	Redundancy
Administration	Local	Remote & Local	Local
Project Status	Popularly available	Already released	Development Stage
Session maintenance	Connection based	User based	N/A
Redirection of request	Once	Once	Multiple
Crash Recovery	No failover detection & no recovery	Failover support for SQL databases and file & print services	Failover detection available but no recovery
Number of system in a cluster	No limit	Maximum 32 servers	Demonstrated for up to 72 hosts

Table 2.1: Comparison of Native Solutions (Continued Page)

2.3 Application Development Platforms

These are custom development environments like DCOM [URL11, URL12] and J2EE [URL13] enterprise solutions. These solutions provide these features

2.3.1 Session Manager

This is required to keep the sessions of the users, to know the user better and to give him the personalised service. Security purposes so we can guarantee that a not logged user cannot go on for further requests so we can deny him service. Application session maintenance across each request transfers.

2.3.2 Asynchronous Messaging

All cluster nodes are not based on synchronous messaging since a sender will be busy to prepare the previous transfer and the receiver will be busy to execute previously received requests and need not be free. Also after processing all the issues of a command

the result too has to be sent to the client directly that is we need not traverse the same path, so each execute independently.

2.3.3 Process Scheduler

The least loaded system has to be chosen, after that naming convention has to be made to send it to that system. The task is done in a distributed architecture, by having one Process Scheduler for every computer.

2.3.4 Interoperability

The interoperability of using an object type which was generated in a VC++ program is totally different from an object type which is generated in Java program. Hence we need to use industry common data representation format XML [URL1] file format. This needs an XML parser and XML de-parser.

2.3.5 Peer to Peer Technologies

Peer-to-Peer computing is an emerging distributed computing technology that enables direct resource sharing of both computing services and data files among a group of mutually trusted clients over the Internet. JXTA technology is a network programming and computing platform that is designed to solve a number of problems in modern distributed computing, especially in the area broadly referred to as peer-to-peer computing, or peer-to-peer networking, or simply P2P. On 25 April 2001, the first prototype implementation was unveiled on <http://www.jxta.org>. It is present on JDK release 1.1.4 onwards, hence it is present in most common Java platform available on machines running Microsoft Windows and Unix.

2.4 Application servers [URL2,13]

These are single function servers used for dedicated tasks like Web caching/acceleration, Web hosting, networked attached storage, load balancing etc. Appliance servers are designed for quick installation and simple maintenance. These systems come with a pre-loaded operating system and application software (often Web server software) that simplifies deployment so servers can be plugged into networks as easily as desktop PCs.

2.4.1 BEA Architecture [URL3]

BEA Tuxedo is the proven platform that simplifies distributed transaction processing and message-based application development while delivering unlimited scalability and standards-based interoperability. The figure 2.1 shows the BEA Architecture for the application server.

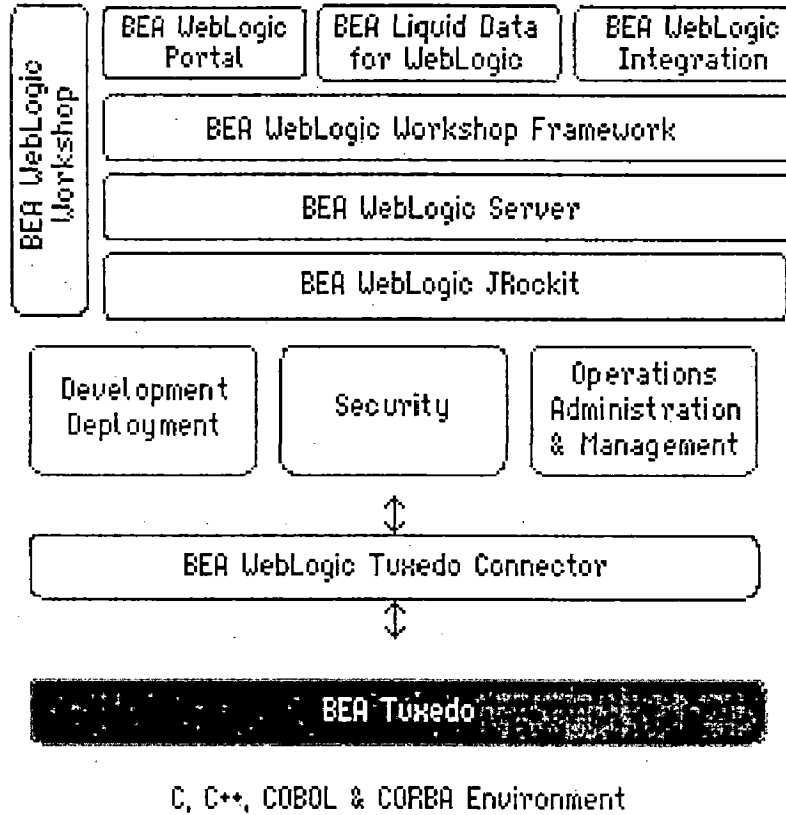


Figure 2.1: BEA Architecture

Key Features and Benefits

The different key features of highly distributed transaction processing in BEA and their Benefits are listed in table 2.2.

Feature	Benefit
Distributed transaction management	Optimizes transactions across one or more databases and ensures data integrity across all participating resources, regardless of the access protocol

Table 2.2: Highly Distributed Transaction Processing

Two-phase commit	Automatically tracks transaction participants and ensures that all databases are updated properly, or will “roll-back”, assuring data integrity despite component failures
Multiple messaging protocols	Supplies synchronous, asynchronous, and conversational messaging APIs for heterogeneous platform support
Transaction queuing	Provides flexibility in processing or deferring transactions to allow distributed applications to work together in an asynchronous, “connection-less” fashion
Event brokering	Provides a transactional event system based on the publish-and-subscribe programming model

Table 2.2: Highly Distributed Transaction Processing (Continued Page)

The different key features of resource management in BEA and their Benefits are listed in table 2.3.

Feature	Benefit
Authentication, authorization, and encryption (LLE)	Ensures data privacy when deploying BEA Tuxedo applications across networks
Security plug-in framework	Enables public key encryption, digital signatures, and 3rd party security products integration
Common Object Request Broker Architecture (CORBA)	Allows organizations to leverage their existing investments in legacy applications and enables 3rd party Object Request Brokers to bootstrap and authenticate to BEA Tuxedo CORBA servers

Table 2.3: Resource Management

Application to Transaction Manager Interface (ATMI)	An X/Open API supplies a consistent application-programming interface for C, C++, and COBOL across all BEA Tuxedo platforms
Web Based OA&M GUI	Provides sophisticated application management tools and interfaces into the leading network and system management products that simplify application OA&M

Table 2.3: Resource Management (Continued Page)

The different key features of unlimited scalability and reliability in BEA and their Benefits are listed in table 2.4.

Feature	Benefit
Application parallelization	Enables applications to handle requests in parallel and process multiple transactions simultaneously on different, distributed nodes
Replicated service framework	Dynamically replicates distributed applications throughout the network to maximize performance and reliability
Robust fault management	Minimizes downtime and keeps applications running through planned and unplanned downtime by eliminating single points of failure
Automated load management and balancing	Provides automated service replication based on real-time system loads and dynamically balances requests across all available resources ensuring consistently high throughput
Data dependent routing	Routes messages based on their context, content, or time of day and enables efficient transaction processing and prioritization

Table 2.4: unlimited scalability and reliability

Advanced message queuing paradigm	Delivers a flexible, "in-memory" message queuing mechanism for high performance, reliable, asynchronous message delivery
-----------------------------------	--

Table 2.4: unlimited scalability and reliability (Continued Page)

The different key features of extensible infrastructure in BEA and their Benefits are listed in table 2.5.

Feature	Benefit
WebLogic Tuxedo Connector (WTC)	Provides bi-directional, peer-to-peer, cross-platform interoperability with complete transaction and security propagation for data integrity
Interoperability with BEA WebLogic Platform	Streamlines complex business processes thru BEA's best-of-bread J2EE products and leverages existing BEA Tuxedo infrastructure assets
Web services support via BEA WebLogic Workshop and Tuxedo Control	Simplifies Web services generation and deployment with a declarative programming model
Standards-based application integration via BEA WebLogic Integration™	Speeds BEA Tuxedo application integration with new and existing solutions, streamlining complex business processes and connectivity with business partners
XML buffer and parsing support	Supports XML message parsing and routing to other XML-capable applications (i.e. Oracle 9i or to BEA eLink).

Table 2.5: Extensible Infrastructure

2.4.2 Pramati Server [URL2]

Pramati Server is also one of the application server models. It is very easy to configure and have rich set of features. The drawback is their cost, which ranges from 7 lakhs per system installation.

Key Features in Pramati Application Server

1. Drag and drop applications into deploy "basket" directory. Server auto-generates missing XMLs in archives.
2. Point and run applications "As They Are", on Tomcat Server and Apache HTTP Server. No change in directories or files.
3. Personalize shell commands by using the Extensible Pramati Server Command Shell to define commands you want.
4. Smart web load balancer, filtering requests to nodes based on sessions, URLs, availability and workload.
5. Dynamic content cache turbo-charges application performance deployed on Pramati Server. Choose what and when to cache.
6. In-depth statistics and on-the-fly graphs highlight all parameters that show performance and workload.
7. Drill-down diagnostics shrinks resolution time by separating platform issues from application problems.
8. Extensive logging of J2EE server and web activity, makes for clearer and faster reporting of problems.
9. High-availability data source configuration and capability to use multiple driver versions.
10. Customize the server to exactly fit your application, by switching on only the required services, directly in XML.

2.4.3 On Demand Distributed Computing With WebSphere

Before applications can move into production, enterprises face the inevitable and time consuming question of what infrastructure and underlying capacity will be needed to support their applications. This capacity-planning picture isn't getting any prettier either. As all business applications are increasingly Internet-enabled and now available to a

global user base, this becomes more of guess estimation than exact science. On Demand Computing offers the promise of providing computing cycles on a pay per use basis much like electric or gas utilities. The ability to scale "On Demand" is great but it doesn't guarantee performance. This is especially true for Internet applications. Because utility computing doesn't necessarily place resources close to requesting users, it still forces all requests to a central point for processing. To successfully support Web-based applications and avoid the Internet bottlenecks inherent with "silo" serving, a distributed computing model is required.

Using On Demand Distributed Computing, applications not only scale on demand—they avoid the inherent bottlenecks on the Internet. With a few adjustments in application development and design, businesses can propel applications into production without spinning cycles on costly infrastructure decisions. Most importantly, the On Demand Distributed Computing model boosts performance so that applications are never "dumbed-down" to handle the vagaries of the Internet. Developers gain the freedom to create innovative applications in far less time than it is possible to do so with traditional solutions. We will see an executive level overview of how IBM WebSphere applications are easily deployed in the On Demand Distributed Computing model. These practices involve usage of the existing set of services available in Java 2 Enterprise Edition (J2EE) application server containers. In many instances, applications are already viable for On Demand Distributed Computing. In others, following a few J2EE best practices has applications ready for deployment in short order.

2.4.3.1 On Demand Computing Overview

A growing number of businesses are beginning to adopt the On Demand or Utility Computing model as espoused by IBM. In this approach, enterprises *pay only for the computing cycles consumed* instead of paying for infrastructure that has been built to weather periods of peak demand. This economic model is well understood in other industries and is just now penetrating IT. But as is true with any new technology, understanding when and how this architecture can be leveraged is a challenge. In theory, On Demand Computing offers the promise of providing computing cycles on a pay-per-use basis, much like telephone, electric or gas utilities. This ability to scale "On Demand"

sounds great, but it doesn't guarantee application performance. This is especially true for the growing number of Internet applications being deployed daily. Because many On Demand or utility computing architectures are centralized, they still force all requests to a central point for processing. To successfully support Web-based applications and avoid the Internet bottlenecks inherent with "silo" serving, a distributed computing model is required.

Here's why:

The infrastructure used to deliver Web applications typically includes a wide range of technologies including load balancers, HTTP Web servers, caching servers, messaging systems, transaction-processing monitors, application servers, and databases. A typical enterprise application infrastructure is shown in Figure 2.2.

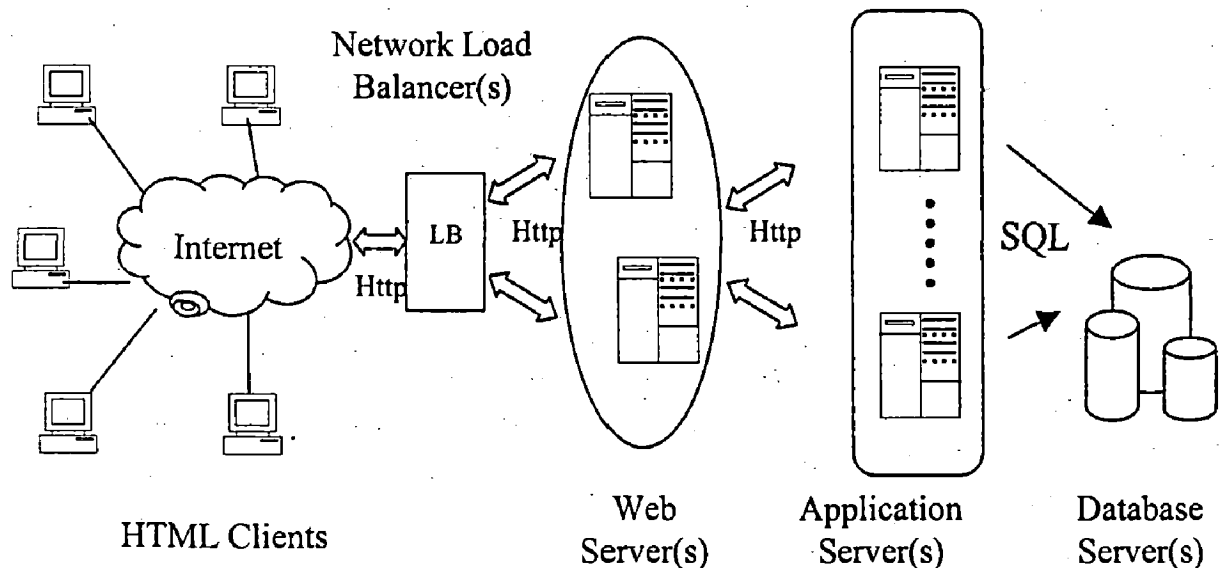


Figure 2.2: Typical J2EE Internet Infrastructure

As performance and geographic reach requirements expand, Internet application infrastructure becomes increasingly heterogeneous and difficult to scale. IT managers continually must evaluate capacity plans to keep pace with the expected peak demand, and the number of "moving parts" increases points of failure. Pre-provisioning extra capacity as insurance against overload is financially unacceptable for most enterprises. In addition, it prevents innovation, as every new application requires a business case proving its worth. Ideally, enterprises want computing resources when—and only

when—they are needed; they do not want to buy extra infrastructure that sits idle when not needed. As seen in Table 2.6, below, a recent study conducted by IBM, demonstrates that Intel and Unix servers deliver sub-10% utilization rates.

	Peak-hour Utilization	Prime-Shift Utilization	24-hour Period Utilization
Mainframes	85-100%	70%	60%
Unix	50-70%	10-15%	<10%
Intel-based	30%	5-10%	2-5%
Storage	N/A	N/A	52%

Source: IBM Scorpion Whitepaper: Simplifying the IT Infrastructure, 2002

Table 2.6: Typical Infrastructure Utilization

“On Demand” computing provides better utilization of computing resources and represents a model in which computing resources are brought into service as needed.

2.4.3.2 On Demand Distributed Computing (ODDC)

Rather than focus on server-side technologies that tend to deliver marginal performance fixes for Internet applications, enterprises can design applications such that they leverage the benefits of a distributed computing platform that extends beyond the server. In general, there are two ways in which On Demand Computing is being made available to businesses: through centralized and distributed architectures. Centralized approaches to utility computing are ideal for applications that reside on a LAN or in instances where user distribution is very low. With the growing number of Web Services and Internet-enabled applications, this isn't always the case, as users are typically spread across the Internet. In contrast to centralized approaches, On Demand Distributed Computing places processing close to users so that application performance and reliability are improved. Businesses that factor this model into design stages are able to launch applications without the normal delays of infrastructure capacity planning. Internet-based applications move from pilot to production more quickly and inherently have higher performance and reliability.

2.4.3.3 ODDC and IBM WebSphere Application Server

Akamai's EdgeComputing service is a leading example of an On Demand Distributed Computing platform. Deployed at the "edge" of the network—close to users' access points—EdgeComputing consists of tens of thousands of servers in over 1,100 networks around the world. J2EE application processing is distributed across this platform so that "On Demand" computing is available close to requesting users. These globally distributed edge servers implement industry standard protocols to support tasks as simple as page assembly or as complex as J2EE processing.

To further extend distributed computing capabilities beyond the presentation tier, Akamai supports the IBM WebSphere Application Server (WAS), Version 5.0, throughout this distributed computing platform. This service—Akamai EdgeComputing powered by WebSphere, enables enterprises to run J2EE Web tier applications in an On Demand WebSphere environment and to consume Internet computing resources on a pay-as-you-go basis. A typical Akamai EdgeComputing environment, consists of the end user using a browser, the enterprise (origin) running business logic, legacy systems and databases, and a distributed network of servers running an embedded WAS or Tomcat server that supports the J2EE web application programming model.

The EdgeComputing development model remains standard J2EE and does not require the use of any proprietary APIs; it is the deployment model that changes, not the programming model. If applications generally follow J2EE component programming best practices, adapting the existing application for EdgeComputing will be easier. EdgeComputing extends the WebSphere application-programming platform to enable the execution of J2EE web tier application components—JSPs, Servlets, Tag libraries, and JavaBeans.

Development for EdgeComputing still relies on standard J2EE development tools and best practices in developing applications, but one must architect edge-enabled applications as two cooperating sub-applications: an edge-side application running on EdgeComputing Platform and an enterprise-side application.

The beauty of this approach is that many enterprises that are running J2EE can adopt On Demand Distributed Computing with few, if any, changes to enterprise applications. By using the existing set of services available in J2EE application server

containers, businesses can designate what processing occurs at the “edge” and what is handled at the enterprise origin. In general terms, this means moving what is known as J2EE “Web Container” application components—JSPs, Servlets, Tag libraries, and JavaBeans—to a tier of “edge servers”. These distributed servers field all application requests, process the Web Container components, and communicate with back-end systems as needed. These requests to back-end systems are handled via industry standard protocols such as HTTP, SOAP, Java RMI (Remote Machine Interface) and Java Database Connectors (JDBC).

EdgeComputing Application Examples

EdgeComputing powered by WebSphere enables a powerful new deployment model for J2EE Web applications. The following examples describe some applications modelled to run on EdgeComputing and illustrate the use of WebSphere Web Services and Cloudscape in EdgeComputing applications.

- Product Catalogue

A product catalogue browsing application can run almost entirely on the edge in the EdgeComputing environment. Since most product catalogues consist of relatively static product data (not including inventory information), the edge application can utilize Cloudscape as the local DBMS. The data can be bundled into the edge WAR (web application archive- packaged j2ee components, server side database beans, shopping cart and static pages) along with the catalogue browsing presentation components. Using this deployment model, it is feasible for the end user browsing interaction to be handled entirely by the edge application. When a user is ready to purchase any selected items, the edge application tunnels back to the enterprise for order processing.

- Marketing Promotional Contest

An enterprise wants to conduct a large-scale marketing promotion to give away a certain new product. Because of the uncertainty of the number of end user contestants, an On Demand edge application is extremely beneficial to assuring a successful outcome. In this scenario, the application might have “random selection” logic to determine if an end user is a winner. An EdgeComputing application can be designed and developed to execute this logic on the edge, offloading the load from the enterprise. In addition, the

corporate marketing team can implement various controls on how long the contest runs, how many products are given out, the rate at which they are disbursed, or other controls. The edge application executes the corresponding business logic entirely on the edge and retrieves the control parameters from the enterprise via Web Services calls.

- Site Search

Search is by far the most frequently used application on Internet sites and can consume significant application server resources in terms of request handling and requisite back end queries. When deployed using ODDC, Search applications powered by EdgeComputing uses Cloudscape to store data at the edge and IBM WebSphere to execute searches close to users, thereby offloading the load from the enterprise.

2.4.3.4 Features of Akamai

Cache consistency

When objects that the edge servers deliver are cacheable, we must address the consistency of cached content; when they are uncacheable, high-performance delivery is a challenge. To address cacheable-object consistency, content providers often use established techniques, such as applying a "time to live" (TTL) to objects. Some objects might be cacheable forever, or at least until they are explicitly removed by a cache control utility (for more on this, see the "Lifetime Control" section). Another approach is to use a different URL for each object version. In addition to using a unique query string for this purpose, Akamai let customers place a version or generation number in the URL. Versioned objects typically have infinite TTLs. To improve uncacheable objects' performance, Akamai introduce an edge server between the client and origin to split the client's TCP connection into two separate connections one from the client to the edge server and one from the edge server to the origin. Contrary to intuition, splitting the connection can deliver faster responses in some cases because the edge server can react to packet loss more quickly than the origin server, improving the connection's bandwidth-delay product. Akamai also map clients to edge servers that have low congestion and packet loss. Furthermore, the edge server can accept the origin server's response faster than the client could, and can serve it from memory at the client's pace. This frees up origin server resources to serve subsequent requests, reducing origin site demand even for

uncacheable content. Finally, the edge server can maintain much longer persistent connections with the client than can an origin server; the origin need only maintain connections with relatively few Akamai edge servers.

Lifetime control

In some cases, the edge server must remove certain objects from all servers on demand. This might be in response to a request from an Akamai customer (the content's provider), or initiated by an interface that lets content publishing systems schedule invalidations when content changes. Because most Web objects change infrequently, heuristic caching policies in Web proxies typically hold copies long after they change. Akamai's edge servers support on-demand purges for changed or otherwise invalid content.

Authentication and Authorization

When serving protected content, edge servers must either contain authorization features or relay authentication tokens to the origin server for authorization. In the latter case, the edge server must be careful not to evict the protected content on a request authorization failure. Akamai lets content providers authorize every user request from their own site by passing request headers from our edge servers to their content servers prior to serving each client request. Akamai edge servers can also process authorization tokens that the origin server attaches to the request, thereby avoiding a round trip to the origin server on each request.

Integrity control

A server must ensure that each client request receives the correct response, and also detect when origin servers issue incomplete responses and avoid caching those responses. Edge servers can contain content from many customer origin servers, and it's imperative that they not serve content to the wrong customer regardless of the content's name or how clients access it. Furthermore, a server should detect when cached objects become corrupted (due to disk failure, for example) and re-fetch them if they do. In Akamai's system, they have built a content integrity check feature into our software; prior to serving each block of a response, the server double checks that the content is

associated with the request. This protects the edge server from serving content that was corrupted on disk or confused in memory due to a software error.

Visibility into access patterns

Customers want to see detailed content-access logs. To offer this, Akamai aggregate individual server logs and extract relevant entries for each customer. Log delivery and aggregation involves a significant data flow, however, and collecting and processing all the logs can take time. Some content providers also want real-time delivery information about their site. In this case, Akamai focus on giving customers content delivery rates and client locations, rather than full log details.

2.4.3.5 Summary for On Demand Computing

Before businesses embrace "On Demand", they must realize that this new deployment model does not guarantee Internet application performance. To successfully support Web-based applications and avoid the Internet bottlenecks inherent with "silo" serving, a distributed computing model is required. Using On Demand Distributed Computing, Internet applications can deliver new levels of performance and reliability regardless of user location or load. By using the existing set of services available in J2EE application.

2.4.4 Turboworx

TurboWorx provides the world's only fully integrated, end-to-end solution for creating, managing, and accelerating technical computing applications, workflows and data processing in heterogeneous distributed computing environments, including clusters and grids. TurboWorx's SmartGrid technology is built on an open development environment and is available for all major operating systems.

TurboWorx's suite of products addresses these problems. The flexibility of TurboWorx solutions facilitates an ongoing process of experimentation and improvement - attributes not present in the usual solutions built on traditional scripts and batch processing queues. As a result, scientists, engineers, analysts and others can solve complex computing problems with reusable workflows, which become valuable assets to their businesses.

ANALYSIS

In this chapter we will go through the issues related to concepts of different parallel computing models, scheduling algorithms, load sharing, scalability and inter node communication.

As Web sites become popular, they are increasingly vulnerable to the flash crowd problem, in which request load overwhelms some aspect of the site's infrastructure, such as the front-end Web server, network equipment, or bandwidth, or (in more advanced sites) the back-end transaction-processing infrastructure. The resulting overload can crash a site or cause unusually high response times both of which can translate into lost revenue or negative customer attitudes toward a product or brand. This requires using a high performance than the average load performance systems.

3.1 High End Computing

In a supercomputing facility that hosts high-performance servers, users can submit various applications, data- or processor-intensive (or both). Users can supply their own software and data or use the locally available software on their respective data. Thus, the type of applications executed can vary widely and, consequently, so can the respective applications' computation times.

3.1.1 Multiprocessor Systems: (Multiple Processors in a Single System)

Multiple processors were once the exclusive domain of mainframes and high-end servers. Today, they are common in all kinds of systems, including high-end PCs and workstations. The most common architecture used in these devices is symmetrical multiprocessing (SMP). The term 'symmetrical' is both important and misleading. Multiple processors are, by definition, symmetrical if any of them can execute any given function.

This point might seem hardly worth emphasizing, but when multiprocessing models first appeared, some were not symmetrical. On these systems, one or more

processors were dedicated to certain specific functions—generally, running the operating system or one of its subsystems. These processors could not run user code and so the design was not symmetrical, since it was not true that any given task could run on any processor.

3.1.2 Massively Parallel Systems: (Collaborating without Shared Resources)

Massively parallel processing (MPP) takes the concepts multiprocessor systems and expands them in a different direction. MPP systems use hundreds of processors, each one supported by its own memory and its own copy of the operating system. Each of these independent computing units is called a node. Nodes share information over a custom high-speed interconnects. MPP systems differ from the systems described previously because all nodes are working under the control of a single program.

For MPP computation to work correctly, the software has to be capable of partitioning its work and the data it operates on over hundreds of processors. This requirement necessitates specialized skills and uncommon programming tools and techniques. MPP is used in scientific applications and in advanced business contexts such as data warehousing and decision support. In both of these business applications, chunks of data are analyzed separately and the results are later aggregated—an almost perfect match for MPP-style computing.

3.1.3 Clusters: (Aggregating Machines into a Single System)

A cluster is a group of individual, stand-alone computers that work together and that outside systems view as a single computing resource. The individual systems (nodes) that make up the cluster communicate with each other via high-speed connections such as Gigabit Ethernet, ATM, or a proprietary link. For easier management, clusters use special software to coordinate and manage their activities, depending on how they are used. Clusters are particularly well suited to meeting the needs of high-availability, load balancing and scientific computing.

- **Highly available clusters** consist of two or more nodes that are exact mirror images of each other. If the primary system goes down due to hardware malfunction, for example, fail-over software immediately makes its twin system the primary node. This approach enables work to continue without interruption.

- **Load-balancing clusters** process heavy volumes of transactions of a similar type. For example, enterprises often use clusters for hosting Web servers or handling database transactions. The cluster routes the incoming transaction stream to whichever node in the system is most able to handle it. Sometimes this decision is based on workload and sometimes it is based on other factors.

3.1.4 Grids: (Resource Sharing among Separate, Distinct Systems)

While clusters are groups of computers tied together as a single device, grids consist of multiple systems that work together while maintaining their distinct identities

This model already has been demonstrated in the wider community of users with the SETI project (among others), which used the home PCs of volunteers to perform analysis of astronomical data

Term computational grid comes from an analogy with the electric power grid:

- Electric power is ubiquitous
- Don't need to know the source (transformer, generator) of the power or the power company that serves it

3.2 Scheduling Algorithms

Scheduling algorithm is used to distribute traffic among the cluster nodes. We will go through some of the common scheduling algorithms and their characteristics.

3.2.1 Round-Robin Scheduling

Distributes each request sequentially around the pool of real servers. Using this algorithm, all the real servers are treated as equals without regard to capacity or load. This scheduling model resembles round-robin DNS but is more granular due to the fact that it is network-connection based and not host-based.

3.2.2 Weighted Round-Robin Scheduling

Distributes each request sequentially around the pool of real servers but gives more jobs to servers with greater capacity. Capacity is indicated by a user-assigned weight factor, which is then adjusted upward or downward by dynamic load information.

Weighted round-robin scheduling is a preferred choice if there are significant differences in the capacity of real servers in the pool. However, if the request load varies

dramatically, the more heavily weighted server may answer more than its share of requests.

3.2.3 Least-Connection

Distributes more requests to real servers with fewer active connections. Because it keeps track of live connections to the real servers through the tables, least-connection is a type of dynamic scheduling algorithm, making it a better choice if there is a high degree of variation in the request load. It is best suited for a real server pool where each member node has roughly the same capacity. If a group of servers have different capabilities, weighted least-connection scheduling is a better choice.

3.2.4 Weighted Least-Connections

Distributes more requests to servers with fewer active connections relative to their capacities. Capacity is indicated by a user-assigned weight, which is then adjusted upward or downward by dynamic load information. The addition of weighting makes this algorithm ideal when the real server pool contains hardware of varying capacity.

3.2.5 Locality-Based Least-Connection Scheduling

Distributes more requests to servers with fewer active connections relative to their destination IPs. This algorithm is designed for use in a proxy-cache server cluster. It routes the packets for an IP address to the server for that address unless that server is above its capacity and has a server in its half load, in which case it assigns the IP address to the least loaded real server.

3.2.6 Locality-Based Least-Connection Scheduling with Replication Scheduling

Distributes more requests to servers with fewer active connections relative to their destination IPs. This algorithm is also designed for use in a proxy-cache server cluster. It differs from Locality-Based Least-Connection Scheduling by mapping the target IP address to a subset of real server nodes. Requests are then routed to the server in this subset with the lowest number of connections. If all the nodes for the destination IP are above capacity, it replicates a new server for that destination IP address by adding the real server with the least connections from the overall pool of real servers to the subset of

real servers for that destination IP. The most loaded node is then dropped from the real server subset to prevent over-replication.

3.2.7 Destination Hash Scheduling

Distributes requests to the pool of real servers by looking up the destination IP in a static hash table. This algorithm is designed for use in a proxy-cache server cluster.

3.2.8 Source Hash Scheduling

Distributes requests to the pool of real servers by looking up the source IP in a static hash table. This algorithm is designated for routers with multiple firewalls.

3.3 Characteristic of a node

Any node in the perspective of clustering can have these characteristic defined as being nearest, being available, being likely.

1. **Nearest** is a function of network topology and dynamic link characteristics: A server with a lower round-trip time is considered nearer than one with a higher round-trip time. Likewise, a server with low packet loss to the client is nearer than one with high packet loss. The design is covered in master/local servers' topic in chapter 4.
2. **Available** is a function of load and network bandwidth: A server carrying too much load or a data center serving near its bandwidth capacity is unavailable to serve more clients. The 100% utilization is not optimum as we study from queuing theory. Hence the design is covered in Setting upper limit topic in chapter 4.
3. **Likely** is a function of which servers carry the content for each customer in a data center: If all servers served all the content by round-robin DNS, for example then the servers' disk and memory resources would be consumed by the most popular set of objects. This is explained in more broader way in the Load sharing issues topic.

3.4 Load sharing issue

LOAD sharing provides a system mechanism to dynamically migrate jobs from heavily loaded workstations to lightly loaded workstations, aiming at fully utilizing system resources. Following the load sharing principle, researchers have designed different alternatives by balancing the number of jobs/tasks among the workstations (see,

e.g., [7], [8]), by considering memory allocation requirements of jobs (see, e.g., [5], [6]) and by considering both CPU and memory resources (see, e.g., [3], [4], [2]). In a cluster system with dynamic load sharing support, a new job can be submitted to a workstation or a running job can be migrated to the workstation under following conditions. When the workstation has idle memory space, the job can be accepted if the number of running jobs in the workstation is still less than a predetermined threshold which is the maximum number of job slots a CPU is willing to take (also called the CPU Threshold). When the workstation does not have idle memory space, or is even oversized, no jobs will be accepted without further checking the status of the CPU threshold.

However, in such a system, a small number of running jobs with large memory allocation requirements can be scattered among workstations to quickly use up the memory space, impeding job submissions to these workstations. Since these large jobs normally have long remaining processing times, eventually, all the workstations may become heavily loaded, stopping job submissions and migrations. We can call this phenomenon the job-blocking problem, which is rooted from unsuitable placements of these large jobs. The existence of these large jobs in a few workstations may increase the queuing delay times of the rest of jobs with relatively small memory requirements, slowing down executions of individual jobs and decreasing the cluster system's throughput. Since job sizes including the memory allocations are unknown in advance, the possibility of unsuitable job placements to cause the blocking problem is high and existing load sharing schemes are unable to effectively handle this problem.

When both job submissions and migrations are blocked in a cluster, it implies that the resource allocation in each workstation either reaches its memory threshold due to arrivals of some jobs with large memory demands, or reaches its CPU threshold, or both. Further job submissions or migrations will cause more page faults or queuing delays in a destination workstation. One simple solution would be to temporarily suspend the large jobs so that the job submissions will not be blocked. However, this approach will not be fair to the large jobs that may starve if job submissions continue to flow, or that can be executed only when the cluster becomes lightly loaded. It is observed that CPU and memory resources are actually not fully utilized during the period of blocking [1]. For example, some workstations reaching their CPU thresholds may still have idle memory

space, while some workstations experiencing page faults may still have additional job slots available.

Recent experiments show that, when a cluster system is not able to further accept or migrate jobs, there are still large accumulated idle memory space volumes available among the workstations. This is because demanded memory allocations of a handful of jobs could not fit in any single workstation with other running jobs. It is also found that jobs are not evenly distributed among workstations, which increases the total job queuing time.

This problem is solved out in limited resource allocation design in Chapter 4.

3.5 System Scalability

Network must scale to support many geographically distributed servers and many customers with differing needs. This presents the following challenges.

- Monitoring and controlling of widely distributed servers, while keeping monitoring bandwidth to a minimum.
- Monitoring network conditions across and between locations, aggregating that information. Success here depends on minimizing the overhead added to avoid long lookup times.
- Isolating customers so they cannot negatively affect each other.
- Collecting logs with information about user requests

3.6 Flexibility vs. Performance in communication protocol

High-performance software communication approaches have increased the need for flexible and high-performance communication systems. When trying to reap the well-known benefits of these approaches, the question of what communication infrastructure should be used to link the various components arises. In this context, flexibility and high-performance seem to be incompatible goals. Traditional HPC-style communication libraries, such as MPI, offer good performance, but are not intended for loosely-coupled systems. Object- and metadata-based approaches like XML offer the needed plug-and-play flexibility, but with significantly lower performance. We observe that the flexibility and baseline performance of data exchange systems are strongly determined by their wire formats, or by how they represent data for transmission in heterogeneous environments.

3.7 Peer-to-Peer (P2P) computing

There are several reasons why peer-to-peer computing model is demanding [11]. One of the reasons is that the increasing amounts of Web content and bandwidth among clients will get under-utilised if client/server models continue to be dominant in the Internet, such as the centralized Web search engines and Web servers. Similarly, if every client has to be served by a proxy on a miss, the number of clients connected to the proxy will be limited (nonscalable) and the available bandwidths among the clients will be under-utilised. P2P systems can be classified into two classes: a pure P2P, where peers share data without a centralized coordination and a hybrid P2P, where some operations are intentionally centralized, such as indexing of peers' files.

DESIGN

4.1 Interaction System

The interaction system takes care of the way data is transmitted within system and how data is transmitted from outside to inside of the system. These two cases are totally diverse situations. When data is transmitted within system it is in a secure, less crash prone, known (limited diversity) nodes. When data is transmitted from outside the system to inside the system the data starts from client passes through internet cloud and reaches our system, thus it is error prone, insecure path, client server scenario, unlimited ways in which the interaction occurs. The client till the system interaction is not considered here. We start looking at the system on receiving a neutral data format, from outside the system, how we go on within the system.

4.1.1 Messages

Messages are designed to be usable on top of asynchronous, reliable and unidirectional transport. Therefore, a message is designed as a pipe, containing an envelope and a stack of protocol headers with bodies. The envelope contains a header, the source endpoint and the destination endpoint. An endpoint is a logical destination, given in the form of a URL, on any networking transport capable of sending and receiving stream-style messages. Endpoints are typically mapped to physical addresses by a messaging layer. Such a message format is designed to support multiple transport standards.

Each protocol body contains a variable number of bytes and one or more identity of the sender to the receiver. For example, a message body may be encoded, with the header providing further information on how to decode the content.

4.1.2 Interoperability with XML

Each programming language contains objects with different representation in memory. For us to integrate between them we need to convert our requests into a standard representation we are using XML file [URL1] format and then transmit using industry standard network protocol TCP/IP. This needs XML parser and de-parser.

We need to make a generic parser and de-parser, since we are not aware of the next request we might receive. But, each request type has a different data type and different number of data members hence we will implement the schema validation per object basis. We have solved it by having first few bytes in the header as reference and then using it we will use the appropriate parser or de-parser to convert from XML file.

The reverse direction from programming language Objects to XML file, we use polymorphism to implement in a generic way.

4.1.2.1 Flexibility vs. Performance in communication protocol

After examining the performance implications of using a number of different format, I use valid XML having strict in the sequence of data, hence flexibility and human debugging is also permitted, while the performance is still maintained high.

4.1.3 Inter system peer-to-peer communication

Among the server hosts, peer-to-peer communication has the communicating systems to have equal priority over one another and a system can with full authority to send the data. This violated the traditional client server technology where the systems will have client to originate request and server to give response after this. Here the origination of connection does not mostly matter except the server socket is invoked to show willingness to communicate.

The transmitting thread & receiving thread for each of the host systems, processing thread should work independently for each connection [12]. Hence we go for an asynchronous communication model, where in the transmitting system will provide the transmitting unit the sufficient information to send and keep on processing with next information. The transmitting thread is woken up when it receives a message to transmit and transmits the data and again goes to suspend state. Similarly on the receiving side the

receiving thread waits for information at the input stream and when data come it extracts and puts into the queue for the processing system to process, after putting into the queue this thread goes into wait state. The processing thread after processing the requests can go into suspend mode till the receiving thread gives the data. Hence the system is not under-resourced (no thread has to wait, when data is in the queue) or over-resourced (no thread is polling for data).

4.1.3.1 Peer groups

A peer group is a virtual entity that speaks the set of peer group protocols. Typically, a peer group is a collection of cooperating peers providing a common set of services.

4.1.3.2 Pipes

Pipes are communication channels for sending and receiving messages and they are asynchronous. They are also unidirectional, so there are input pipes and output pipes. Pipes are virtual, in that a pipe's endpoint can be bound to one or more peer endpoints.

A pipe is usually dynamically bound to a peer at runtime via the Pipe Binding Protocol. This also implies that a pipe can be moved around and bound to different peers at different times. This is useful, for example, when a collection of peers together provide a high level of fault tolerance, where a crashed peer may be replaced by a new peer at a different location, with the latter taking over the existing pipe to keep the communication going.

4.1.3.3 Point to Point pipe and Propagating pipe

A *point-to-point pipe* connects exactly two peer endpoints together. The pipe is an output pipe to the sender and input pipe to the receiver, with traffic going in one direction only from the sender to the receiver. A *propagate pipe* connects multiple peer endpoints together, from one output pipe to one or more input pipes. Accordingly, any message sent into the output pipe is sent to all input pipes.

For example when multiple controls a single node has to take then the response and requests should be broadcasted to all the connected system.

4.2 Load Management

Any request to process is considered a load on our system. How we take care of load and how we going to set an upper bound for the load a system can take. We will see these topics

4.2.1 Load balancing and informing

A system might have many applications running and one application can have higher priority which means that application can use more of the CPU time, hence a less priority process can be over loaded but still the system is not overloaded and vice versa. Hence load information is computed per application basis.

The load status varies less frequently also for applications, which take longer time to process each request; the load could be sent with some gap. Informing to the next system is done periodically.

When latest information about load comes, the old information has no value. As the present state of an application is given by the latest load information only, we erase the previous load information when latest load information comes.

4.2.1.1 Process Scheduler

Every request is identified by request type and it is pre-assigned to a processing code. When this request reaches Process Scheduler, it decides where to execute that processing code.

The load balancing system at every system continuously monitors the state of services and their servers and networks. Each of the content servers frequently reports its load to every monitoring application (Process Scheduler), which aggregates and then determines which IP address to return when resolving request. The server can thus shed a fraction of its load when it is experiencing moderate to high load.

To monitor the entire system's health the Application Manager gives work to the CPU and finds the response time. Application Manager uses this information to monitor overall system performance and to automatically detect and suspend problematic servers.

In addition to load balancing metrics, the Application Manager reports loads to centralized server namely LAN Manager.

4.2.2 Load Restriction

There is an upper limit set in every system and the system is monitored whether its upper limit is crossed or not. When the new job comes and this new job made the upper limit to exceed then the new job's thread is given less priority or is abruptly suspended.

4.3 Naming conventions

We assign each Process Scheduler a distinct IP address namely Process Scheduler-indices. We index each message in our workload, for the purposes of identification and addressing, with a distinct positive-integer. Each request type has an index.

4.3.1 Limited resource allocation

With reference to the analysis of load sharing issues, when we have the node being allotted all kinds of jobs, then the node's virtual memory gets full and the page faults starts to occur. Thus making the node to produce large delay. Allotting a node with specific resource solves this problem. And the resources are pre defined in the Application Manager that this node could do these jobs alone. We now are able to restrict the resources in a streamlined way and nodes are utilized in an efficient way.

4.4 Crash Recovery

Crash Recovery is achieved by continuous monitoring for exceptions at the Application Manager and at the LAN Manager side, it is done by the administrator. Also by keeping the logs and when an application fails, then the state of the application at the time of crash is a valuable resource to recover from any such crash [13] in the future. Defect report is prepared from this log by the maintenance team and also to identify the faulty module by the developers.

Crash Avoidance

XML is used for crash avoidance; this avoids miss representation of data.

4.4.1 Using the event log as a data source for crash recovery

You can use the Event Log service to gather information about hardware, software and system problems. Cluster System records events in three types of logs:

- **Application Log.** The Application Log contains events logged by applications or programs. For example, a database program might record a file error in the Application Log. The program developer decides which events to record.
- **System Log.** The System Log contains events logged by the Cluster system components. For example, the failure of a driver or other system component to load during startup is recorded in the System Log. The event types logged by system components are predetermined for the operating system. The event types are,
 - **Error.** A significant problem, such as loss of data or loss of functionality. For example, if a service fails to load during startup, an error is logged.
 - **Warning.** An event that is not necessarily significant, but may indicate a possible future problem. For example, when disk space is low, a warning is logged.
 - **Information.** An event that describes the successful operation of an application, driver, or service. For example, when a network driver loads successfully, an information event is logged.
 - **Success Audit.** An audited security access attempt that succeeds. For example, a user's successful attempt to log on to the system is logged as a Success Audit event.
 - **Failure Audit.** An audited security access attempt that fails. For example, if a user tries to access a network drive and fails, the attempt is logged as a Failure Audit event.

The Event Log service starts automatically when you start Cluster System. By default, security logging is turned off.

4.5 Transparency to application developers

The applications work with a view that it is the only one application is processing that request type. Hence we give to the applications programmer a level of abstraction that he need not worry about the load sharing. But the persistent data need to be stored onto the centralised database as they are not intermediate data.

The applications programmer needs to be abstracted the communication of request to it and the response from it. Thus we have provided request and the communication together.

4.6 Administration

Remote administration is the way of administering an application from an application, which is run as a separate process than the application and it is connected by network. The remote administration sends status messages, error messages, clearing of checkpoints; together with timestamp and the module, which has sent it. The status messages are for saying what each module has done. This is for debugging purpose. The errors are the unexpected situations that happen at runtime of the program.

We send exceptions name, its line of code and the cause of exception to the remote administrator. This is for debugging purpose. Check points are the places in the code if successfully crossed mean the functionality has been successfully executed by our code. These checkpoints are for demo purpose and testing purposes.

Remote administrator needs special information to be transmitted so we need to send less data only. For a human point of view to debug, show demo it is enough we consider OS level processes. So, each OS process has a remote administration-transmitting unit. So that for all the modules the messages is transmitted through the same channel and saying as another tag from differentiates them and which function the message originated.

The location of remote administrator need not be known for the applications hence we have server socket for the applications and the remote administrator connect to this system.

4.6.1 Local administration and remote administration User Interfaces

In local administration the administrator is given provision to individually control a system by using a shell prompt in it. In remote administration the administrator can sit in a remote terminal and manipulate with the controls in every system. By this he will be able to have control over more systems in the network.

4.6.2 Self-managing, Self-organizing, Self-healing

The administration program has a set of rules, which guides it to manage some the troubles and some of the utilities by itself. For example informing about a new system that has been added to the network, to every other system that it knows.

4.7 Design of Modules

We will go through the various class diagrams and their associations and interaction with other modules.

4.7.1 Data Flow

The flow of any request that comes to the Real Time Cluster Management System is given in Figure 4.1. In the diagram, the Firewall interacts with the client and hence forms as User Manager. The Session Manager holds the XML parsing and de-parsing functionality. Process Scheduler distributes the load among the process schedulers residing in another computer.

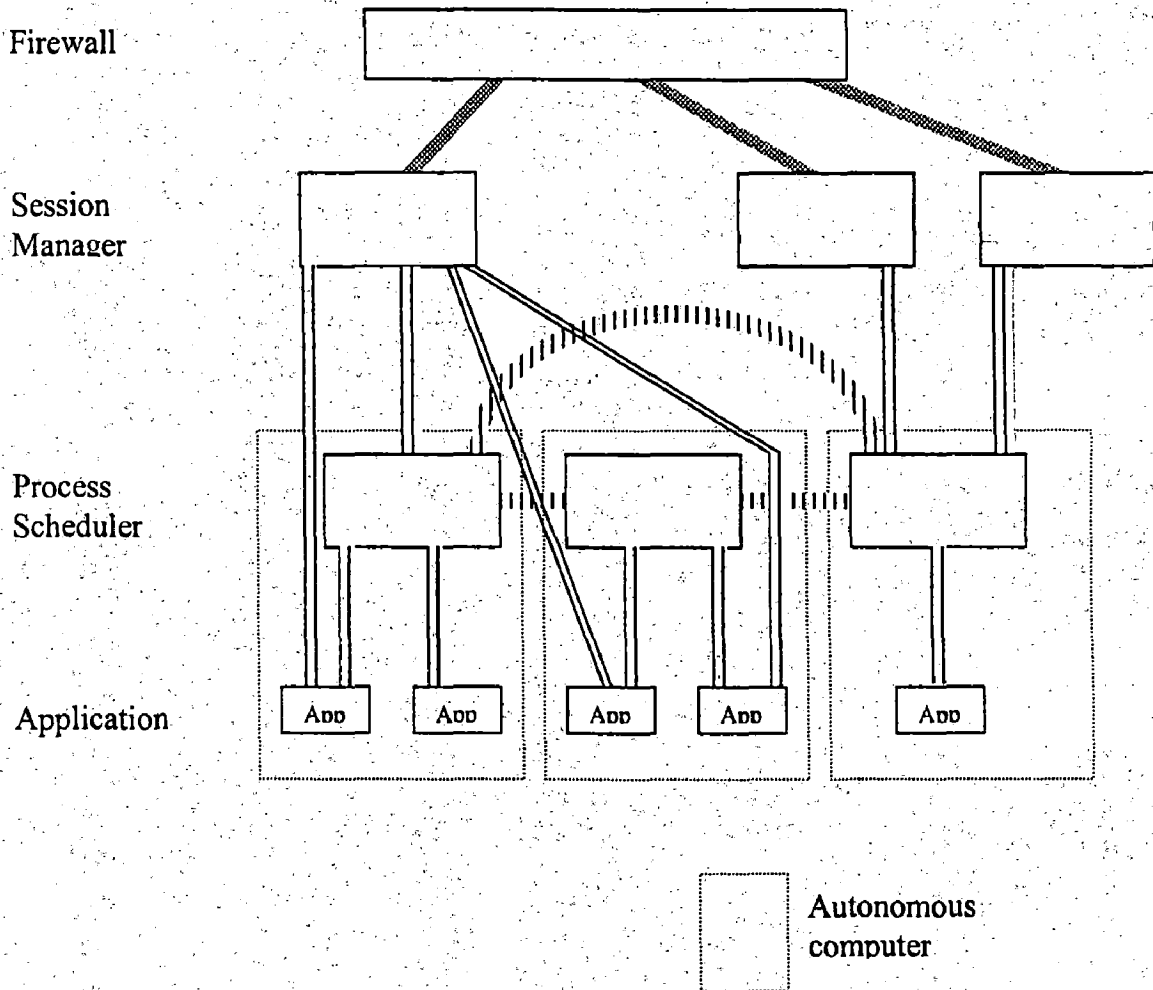


Figure 4.1: Data Flow diagram of Real Time Cluster Management System

4.7.2 Session Manager

The session manager takes care of doing collaborative computing wherein Process Scheduler register at Session Manager, receiving subsequent task to compute at each

requisition and returning the results from that task at a subsequent time. The security of a request is enhanced if the Session Manager can easily keep track of which Process Scheduler computed which tasks, thereby endowing the request with accountability.

We develop a framework for constructing computationally lightweight schemes for endowing requests with accountability. Session manager being the first system for the client, it has to handle requests of type XML file. Session Manager with Firewall, Process Scheduler, and Applications interaction is described in Figure 4.2

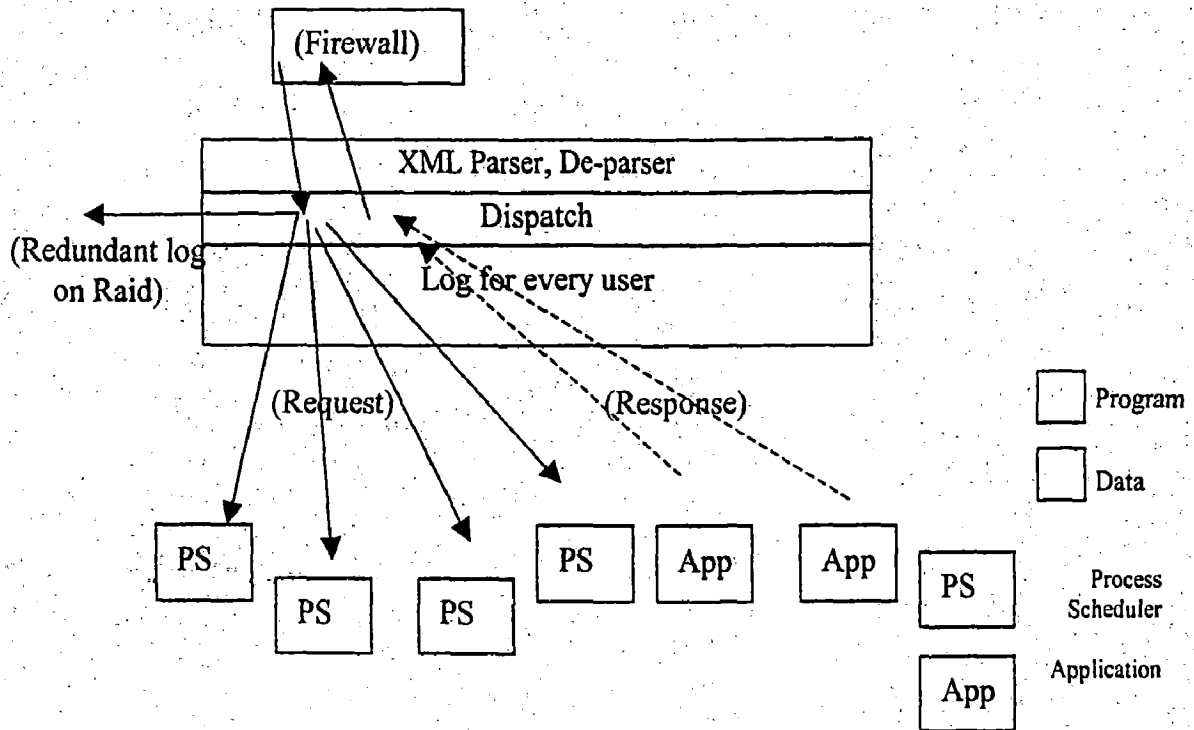


Figure 4.2: Session Manager's interaction with other modules

The Session Manager module's association is as described in Figure 4.3. The FirewallObj is the class instance received or sent with Software Firewall. To transmit the user's request along with the request id that is attached with every request.

ReadObject does asynchronous fetch from Software Firewall. The interaction with firewall is like any other distributed computing interaction but the stream is a bi-directional. SocketByteReader converts the XML file received from Firewall using the XML parser. TransmitterByte converts objects from server applications into XML file using XML de-parser and sends to Firewall.

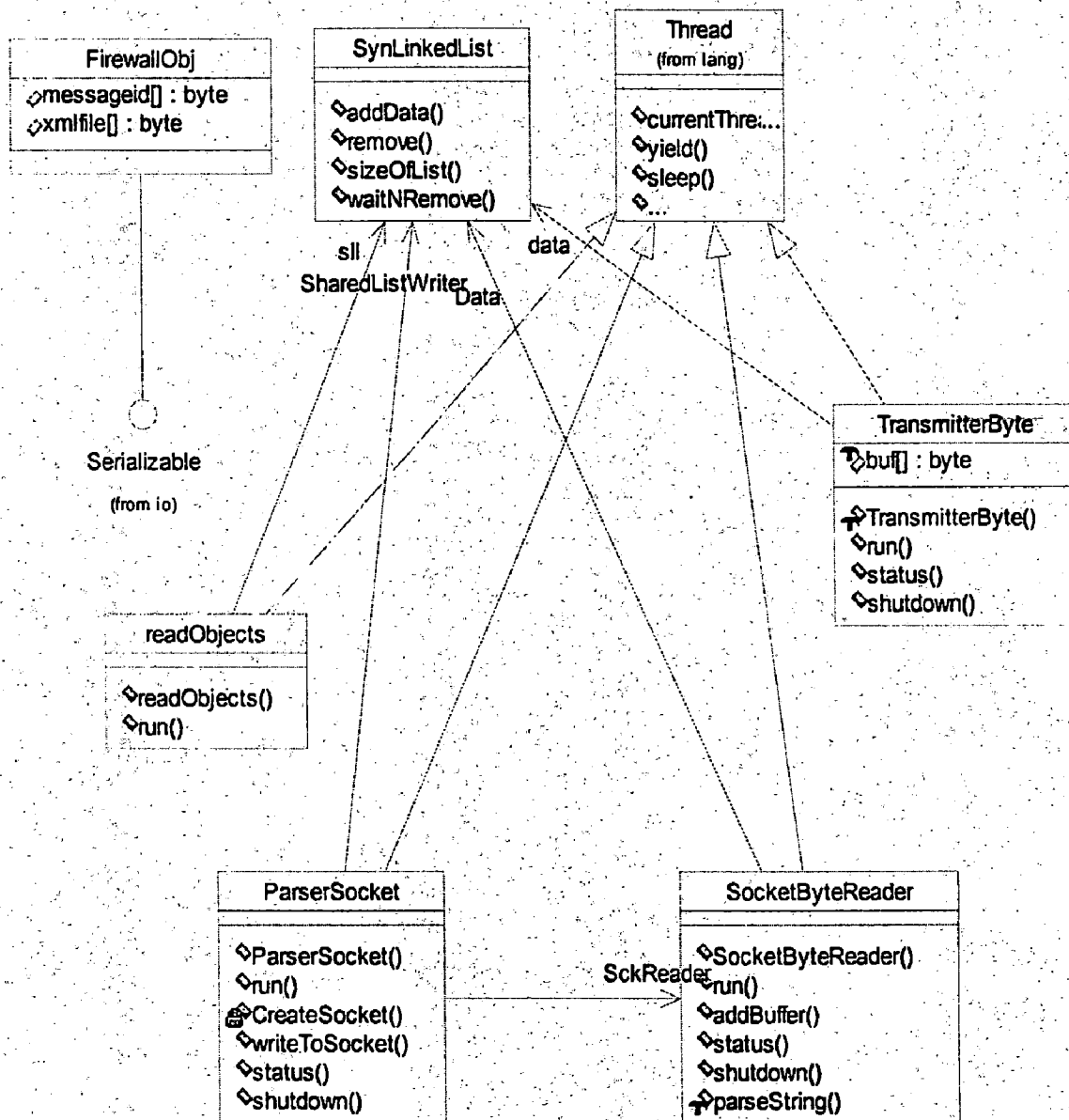


Figure 4.3: Session manager module's associations

4.7.3 GenericSocket interactions

GenericSocket is used to achieve peer-to-peer interaction within internal modules with one another, the GenericSocket class's associations is depicted in Figure 4.4.

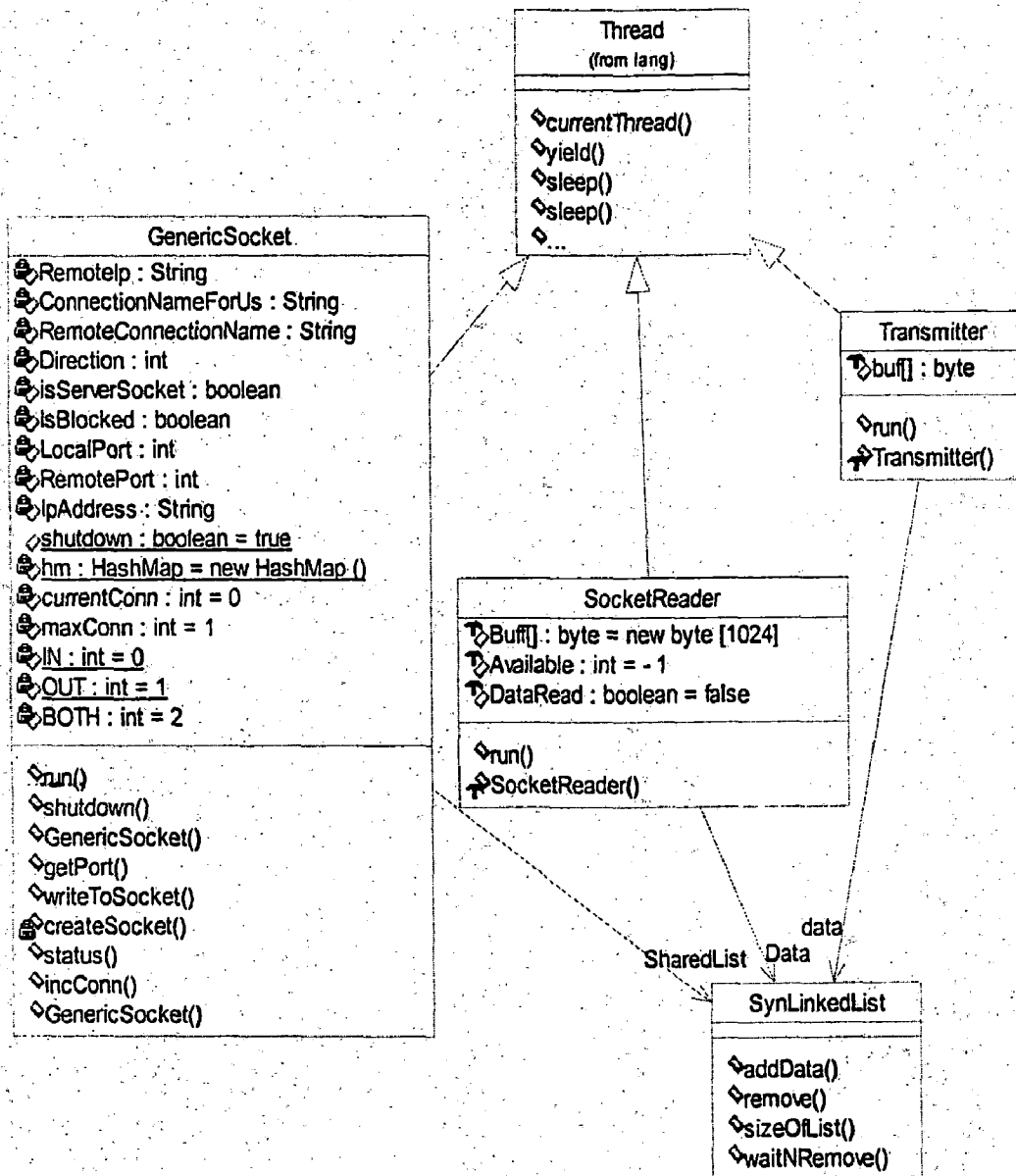


Figure 4.4: GenericSocket class associations

GenericSocket connects to the next system either in client or server socket of TCP/IP. It then invokes SocketReader or transmitter. SocketReader provides reading of objects in a separate thread. Transmitter provider sends objects in a separate thread.

SynLinkedList provides thread synchronisation by making a thread to wait till data comes in and when data comes in it, it wakes the waiting thread.

4.7.4 Process Scheduler

The Process Scheduler is composed of a placeholder that initiates all load balancing features. The load balancing for every request type is taken care by the independent Dispatcher instances, which is specific for every request type. The components of Process Scheduler is described in Figure 4.5.

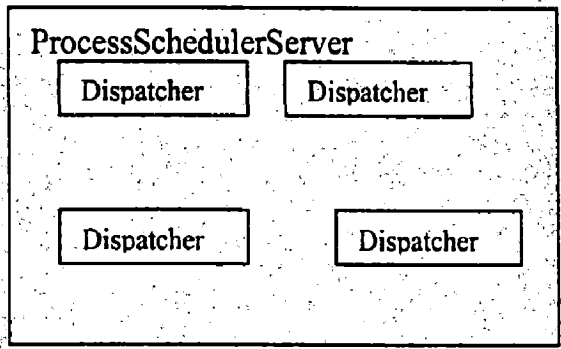
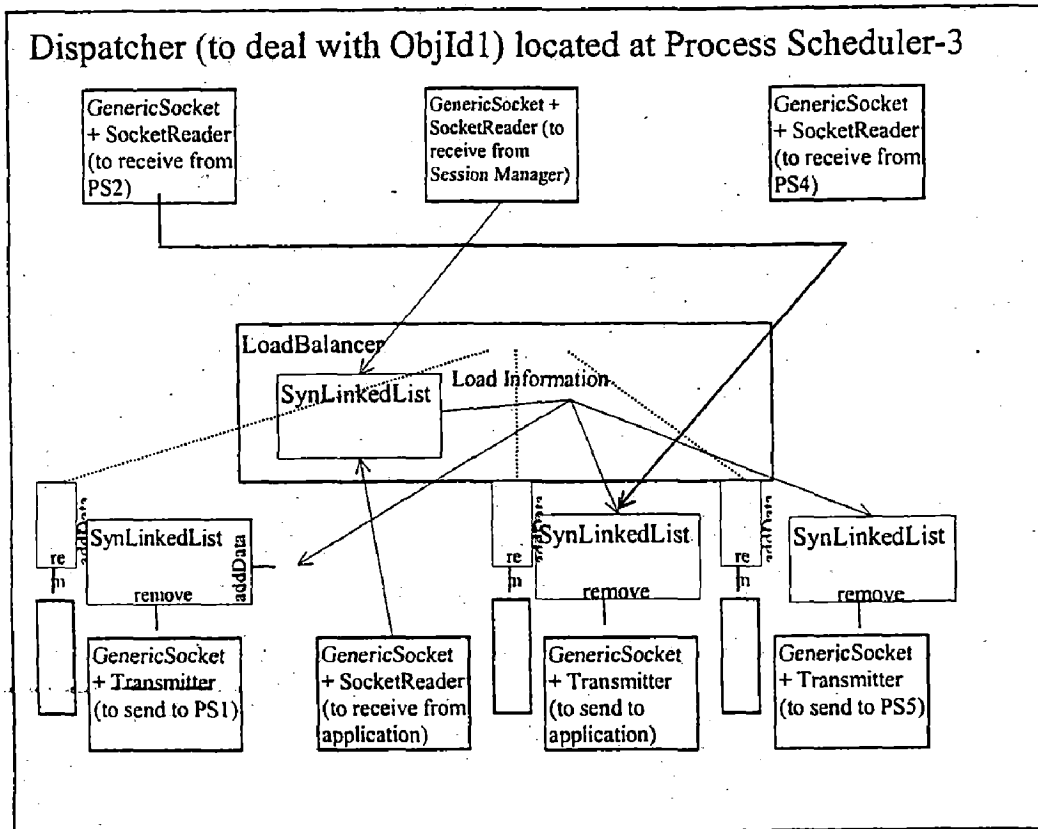


Figure 4.5: Process Scheduler Program co-ordinator

4.7.4.1 Dispatcher

Dispatcher is used to deal with load balancing for a single request type. Hence there will be every instance for each request it receives from session manager and for every instance the local application is going to give. The dispatcher content is described in Figure 4.6.



Reference:

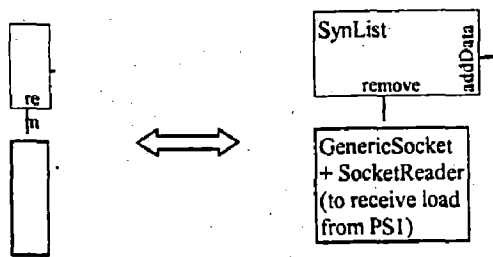
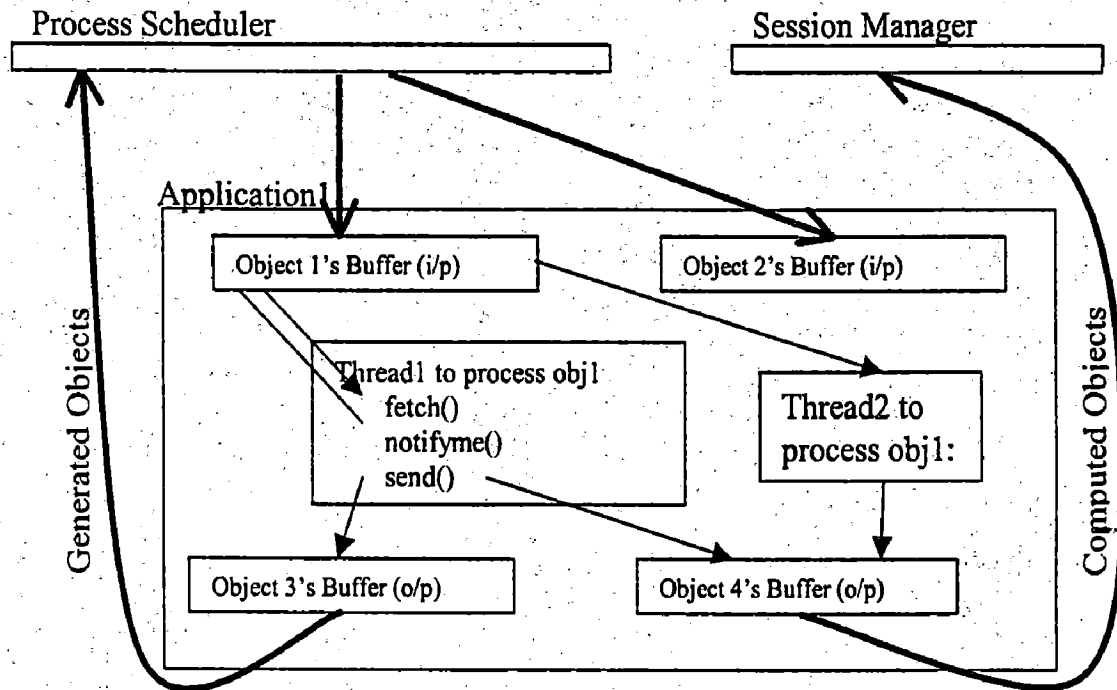


Figure 4.6: Dispatcher instance's content

4.7.5 Wrapper Class for applications

The wrapper class is used to provide application programmers the interface to the cluster. Using the wrapper class an application can send a request to be processed in some place in the cluster, also the receiving of any request is through the wrapper class. The data flow with the wrapper class is shown in Figure 4.7.



Generated Objects- is processed by some Application.
Computed Objects- is sent back to the user

Figure 4.7: Data flow diagram with Wrapper class

4.8 Summary

We have seen, how to solve the issues of interaction system, load management, naming convention, crash recovery and the module design in the code level.

IMPLEMENTATION

5.1 Application Programmer's guide

Create instance of in direction wrapper class for the specific request type this code is able to process. Specify the new request type in the generic Socket class. Then start using the cluster system by calling the function waitNFetch(). This method is thread synchronized and retrieves the data from buffer. If the buffer is empty then the method bring the invoking code to a wait state (Suspend state). When data comes then thread is waken up by the adding code so that this code is able to utilize the data.

Instance of wrapper class is created by calling the appropriate constructor Obj77AppIn(), here 77 is the request id and this is the application wrapper for receiving data. Similarly use Obj77AppOut() for sending data. As per the configuration entered in the GenericSocket class about the request type, the out will send to either client or to any other application for further processing. If it is for further processing then ProcessScheduler decides where to execute the code.

5.2 Administrator's Guide

The administrator use the software to deploy the cluster code and the application on to their site. In this situation the cluster system's remote admin and local administration module help in deploying the required applications in specific computer nodes and they can see the status of the applications that are hosted by using the GUI provided. To send commands to the application is by selecting the computer to give command and choose one of the command that the application provides. Thus we have the latest set of commands to interact with the application. To start any application in a computer select the application manager in that computer and issue the command start applicationName thus that application is started after checking all security and the initial

set-up are done to ensure successful deployment. Then the destination platform is identified and the application is initiated, as it has to be done in that platform.

5.3 Remote Administrator Communication

The remote administrator also communicates through a request type; this request type has the request command, added information, destination node's path and source node's path from the current location.

The shutdown command when it is issued to an application it informs to its direct superior and then shutdown, hence the remote administrator communication path is broken to show the disconnection for that application.

5.4 Marshalling and De-marshalling of data to transmit

When we send an instance object from one system to another it must be converted to byte stream and be transmitted. This is done through XML parsing and deparsing for the transmission between Java and VC++ systems. This requires the object to be marshalled by extending parsing or deparsing interface as per the role. To transmit with in java then java's built in stream is used, namely `ObjectInputStream` and `ObjectOutputStream`. This requires the objects to be transmitted by extending the `Serializable` class.

5.5 Working Environment

The Dissertation was implemented and tested using the following environments,

Operating Systems

1. Windows 2000,XP, 2003.
2. Redhat Linux 8.0 and above versions.
3. Macintosh Jaguar versions.

Programming Language and run time environment

1. Sun Java 1.4.1
2. Visual C++ 6.0 (for sending requests from external source)

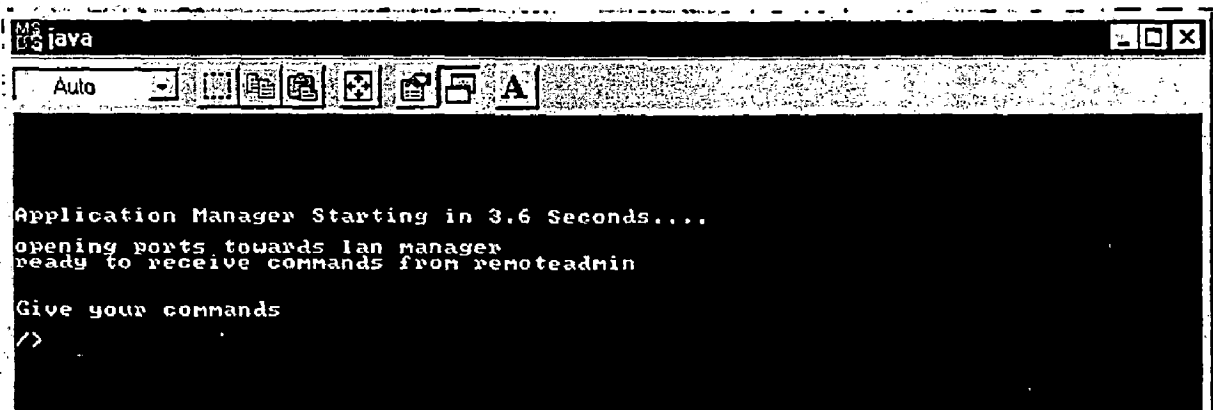
Hardware

1. RAM 1 GB
2. Xeon Processor with 2.6GHz Speed each.

RESULTS AND DISCUSSIONS

6.1 Starting Application Manager

Application Manager is the program used for remote administration purpose used to initiate any other programs. This is started by calling `java remoteadmin.source.AppManager`. Then the commands are typed to start the other applications. Figure 6.1 shows the console screen of the Application Manager being started and is ready to get inputs.



```
Application Manager Starting in 3.6 Seconds...
opening ports towards lan manager
ready to receive commands from remoteadmin

Give your commands
/>
```

Figure 6.1: Starting Application Manager

6.2 Starting programs in a computer

From the application manager's shell window, type the command `start Firewall`, `start ps`, `start sm`. These will start corresponding screens in separate windows. This can be seen in the figure 6.2.

This mode of invoking the `start` command requires an instance of application manager to be in running state in that computer. Thus we need to start an instance of application manager manually and the application manager automatically without the administrator's consent does the other issues.



Figure 6.2 Starting programs

6.2.1 Starting applications

On starting the application the required information is passed between the application by the application manager automatically. The starting of application involves the starting of process scheduler also, because process scheduler dispatches the request to the applications. The application User Management System is started as in Figure 6.2. The command is same, start ums.

6.3 Flow of request

The data flow can be traced from the firewall to session manager to process scheduler and then to application and back to process scheduler and then to any other application and then to session manager and then to firewall to client. This flow can be seen with respect to the request type that is being passed between the cluster nodes as seen in the Figure 6.3

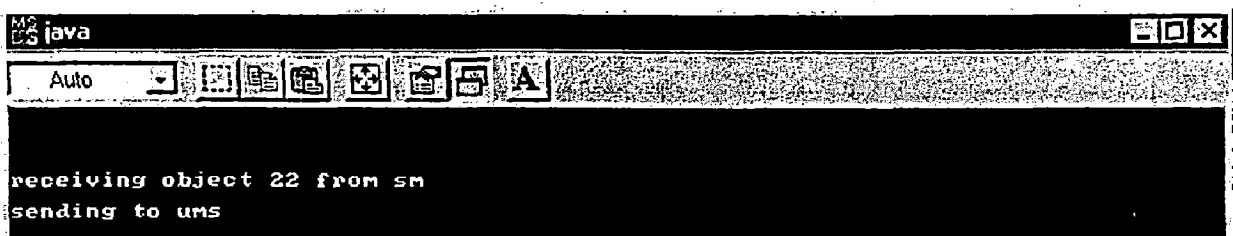
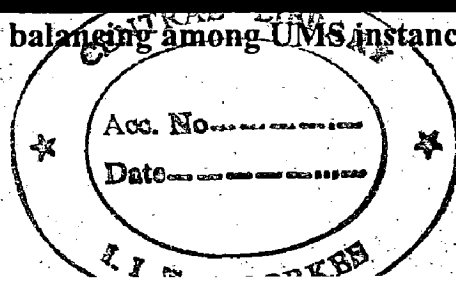


Figure 6.3 Process Scheduler doing load balancing among UMS instances



CONCLUSIONS

Today the growth rate of computing power is very high, but instead of replacing the old systems with new system, the Real Time Cluster Management System helps in successful up gradation of Web Servers. And it avoids the Internet bottlenecks inherent with “silo” serving. Using Real Time Cluster Management System, Internet applications can deliver new levels of performance and reliability regardless of user location or load. The cost was definitely cheaper than the other model, which are less flexible.

This proven model means that enterprises needn't expend time or money on complex capacity forecasting. For developers, the Cluster Management System boosts performance so that applications are never “dumbed-down” to handle the vagaries of the Internet. Enterprises can create innovative applications in far less time than possible with traditional solutions. The distributed architecture has eliminated the single point of failure. And also the crash recovery module has provided a situation to host the server in a continuous working mode round the clock.

Future Scope

Some areas for improvement were identified during the dissertation, which include if the load information is not available for a long time then the load can be assumed to be heavier of so many factors [URL9]. Providing a proxy server capable of handling users with secure encrypted path, between the remote administration servers so that an administrator can connect through a secure path to the internal applications and the internal servers can use another load balancing server from another subnet. Some utility can be made which can roll back the servers to a stable state in the past.

REFERENCES

- [1] L. Xiao, S. Chen and X. Zhang, "Dynamic Cluster Resource Allocations for Jobs with Known and Unknown Memory Demands", IEEE Trans. Parallel and Distributed Systems, vol. 13, no. 3, pp. 223-240, 2002.
- [2] Y. Amir, B. Awerbuch, A. Barak, R. Borgstrom, and A. Keren, "An Opportunity Cost Approach for Job Assignment and Reassignment in a Scalable Computing Cluster", IEEE Trans. Parallel and Distributed Systems, vol. 11, no. 7, pp. 760-768, July 2000.
- [3] L. Xiao, X. Zhang, and S.A. Kubricht, "Incorporating Job Migration and Network RAM to Share Cluster Memory Resources", Proc. Ninth IEEE Int'l Symp. High Performance Distributed Computing, pp. 71-78, Aug. 2000.
- [4] X. Zhang, Y. Qu, and L. Xiao, "Improving Distributed Workload Performance by Sharing Both CPU and Memory Resources", Proc. 20th Int'l Conf. Distributed Computing Systems, pp. 233-241, Apr. 2000.
- [5] A. Barak and A. Braverman, "Memory Ushering in a Scalable Computing Cluster, J. Microprocessors and Microsystems", vol. 22, nos. 3-4, pp. 175-182, Aug. 1998.
- [6] A. Batat and D.G. Feitelson, "Gang Scheduling with Memory Considerations", Proc. 14th Int'l Parallel and Distributed Processing Symp., pp. 109-114, May 2000.
- [7] M. Harchol-Balter and A.B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing", ACM Trans. Computer Systems, vol. 15, no. 3, pp. 253-285, 1997.

- [8] V. Karamcheti and A. Chien, "A Hierarchical Load-Balancing Framework for Dynamic Multithreaded Computations", Proc. Supercomputing Conf., Nov. 1998.
- [9] D.A. Menascé and V.A.F. Almeida , "Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning", Prentice Hall, Upper Saddle River, N.J.,2000.
- [10] CNet News clip, "Pay per Use Computing Cycle", <http://news.com.com/2100-1001-270988.html?legacy=cnet>, dated 01' August 2001.
- [11] L. Gong , "JXTA: A Network Programming Environment, IEEE Internet Computing", vol. 5,no. 3,; May/June 2001.
- [12] Jack Shirazi, "Distributed Computing of book on Java Performance tuning" Chapter. 12, published by O'Reilly press, 2000.
- [13] Y Tohma, "Fault Tolerance", IEEE Distributed Systems Online, February 2004.
- [14] Li Xiao and Xiaodong Zhang, "Adaptive Memory Allocations in Clusters to Handle Unexpectedly Large Data-Intensive Jobs", IEEE Trans. Parallel and Distributed Systems, vol. 15, no. 7, pp. 577-592, July 2004

URLs

- [URL1] XML Task Force, "eXtended Markup Language", <http://www.w3.org/XML/>
- [URL2] Web Application Server, "Specialized in J2EE application hosting Servers", <http://www.pramati.com/index.htm>
- [URL3] Web Application Server, "Specialized in J2EE application hosting Servers", <http://www.bea.com>

- [URL4] Space Research Centre, "Collects computing power from normal pcs to do compute on signals", <http://setiathome.ssl.berkeley.edu/>
- [URL 5] Information on the Windows 2000 Server family, "Configurations and benefits from the windows 2000 Family Servers", <http://www.microsoft.com/windows2000/guide/server>
- [URL6] Rich Farrell, "Distributed Computing", published in Network World on Sep-97' <http://www.nwfusion.com/netresources/0922web2.html>
- [URL7] "Parallel Computing", Published on November 5, 2002 at Techrepublic hosted at <http://techrepublic.com.com/5100-6268-1061235.html>
- [URL8] BED Windows Networking & Communications Team, "Network Load Balancing in Windows2000", Whitepaper published at Microsoft website created by, www.microsoft.com/windows2000/docs/NLBtech2.doc
- [URL9] MOSIX, "University Project on shared memory cluster computing", <http://www.mosix.org>
- [URL10] Java Performance Tuning, "kernel level, OS level, application level Performance tuning", www.javaperformancetuning.com
- [URL11] Microsoft Component Service, "Server Operating System a Technological overview", <http://www.microsoft.com/com/wpaper/compsvcs.asp>
- [URL12] DCOM overview, "Technical tutorial on COM object model", http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp

- [URL13] MTS, "Comparing Microsoft Transaction Server to Enterprise JavaBeans", <http://www.microsoft.com/com/wpaper/mts-ejb.asp>
- [URL14] Crash Recovery, "Crash Recovery in a Distributed Data Storage System", <http://research.microsoft.com/lampson/21-CrashRecovery/Acrobat.pdf>
- [URL15] Application Architecture for .NET, "Designing Applications and Services on Windows Network Load Balancing Cluster", 2000 <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/distapp.asp>