AUTONOMIC PLACEMENT OF HETEROGENEOUS WORKLOADS

A PROJECT REPORT

Submitted by

UMAMAHESWARI S

in partial fulfillment for the requirement of award of the degree of

MASTER OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering

KUMARAGURU COLLEGE OF TECHNOLOGY,

COIMBATORE 641 049

(An Autonomous Institution Affiliated to Anna University, Chennai)

APRIL 2013

AUTONOMIC PLACEMENT OF HETEROGENEOUS WORKLOADS

A PROJECT REPORT

Submitted by

UMAMAHESWARI S

in partial fulfillment for the requirement of award of the degree of

MASTER OF ENGINEERING

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE 641 049

(An Autonomous Institution Affiliated to Anna University, Chennai)

APRIL 2013

BONAFIDE CERTIFICATE

Certified that this project work titled **"AUTONOMIC PLACEMENT OF HETEROGENEOUS WORKLOADS"** is the bonafide work of UMAMAHESWARI S (1120108021) who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other students.

(Signature of the HoD)	(Signature of the Supervisor)
Prof.N.JAYAPATHI M.Tech.,	Ms.D.SATHYA M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Professor,	Assistant Professor,
Department of Computer Science and Engineering,	Department of Computer Science and Engineering,
Kumaraguru College of Technology,	Kumaraguru College of Technology,
Coimbatore- 641 049.	Coimbatore- 641 049.

Submitted for the Project Viva-Voce examination held on ____________.

--------------------------------	-------------------------------
Internal Examiner	**External Examiner**

ACKNOWLEDGEMENT

I express my profound gratitude to **Padmabhusan Arutselvar Dr.N.Mahalingam, B.Sc., F.I.E , Chairman**, **Dr.B.K. Krishnaraj Vanavarayar**, **Co-Chairman, Mr. M. Balasubramaniam, M.Com., M.B.A, Correspondent, Mr.Sankar Vanavarayar**, **M.B.A., PGDIEM, Joint Correspondent** and **Dr.S.Ramachandran Ph.D., Principal** for providing the necessary facilities to complete my thesis.

I take this opportunity to thank **Prof.N.Jayapathi M.E**., Head of the Department, Department of Computer Science and Engineering, for his support and timely motivation. Special thanks to my Project Coordinator **Dr.V.Vanitha Ph.D.**, Senior Associate Professor, Department of Computer Science and Engineering, and project committee members for arranging brain storming project review sessions.

I register my sincere thanks to my guide **Ms.D.Sathya M.E.,** Assistant Professor, Department of Computer Science and Engineering. I am grateful for her support, encouragement and ideas. I would like to convey my honest thanks to all **Teaching** and **Non Teaching Staff** members of the department and my classmates for their support.

I dedicate this project work to my **Parents** for no reasons but feeling from bottom of my heart, without their love this work would not be possible.

- S.UMAMAHESWARI

ABSTRACT

An enterprise data centre consolidates the workload on the same physical hardware in order to reduce the cost of infrastructure and electrical energy. These workloads comprise both transactional and long-running analytic computations. Such consolidation brings new performance management challenges due to the intrinsically different nature of a heterogeneous set of mixed workloads, ranging from scientific simulations to multitier transactional applications. The fact that such different workloads have different nature imposes the need for new scheduling mechanisms to manage collocated heterogeneous sets of applications, such as running a web application and a batch job on same physical server, with differentiated performance goals. Introducing a technique that enables existing middleware to fairly manage mixed workload to permits collocation of the workload types on same physical hardware and perform system reconfiguration.

ஆய்வுச்சுருக்கம்

ஒரு நிறுவன தரவு மைய உள்கட்டமைப்பு மற்றும் மின் ஆற்றல் விலைகளை குறைக்கும் பொருட்டு அதே உடல் வன்பொருள் வேலை பளு ஒருங்கிணைக்கின்றது. பல்வேறு இயந்திரம் செய்யும் வேலைகளை ஒரே இயந்திரம் செய்வதால் அதன் வடிவமைப்பு மற்றும் நேரம் குறைகிறது. வேலைகள் என்பது பணிச்சுமை நடவடிக்கை மற்றும் நீண்ட பகுப்பாய்வு கணக்கீடுகள் ஆகும். ஒரே இயந்திரத்தில் பல வேலைகளை செய்வதால் இயந்திரம் எளிதில் பழுதுடைகிறது. இந்த குறைபாடுகளை நீக்க, வேலைபாடுகளை திட்டமிட்டு மற்றும் இயந்திரத்தின் செயல்பாடுகளை கணக்கிட்டு, வேலைகளை செயல்படுத்த கூடிய இயந்திரத்திற்கு வேலைகள் அனுப்பபடுகிறது. இதனால் அனைத்து இயந்திரமும் செயல்படுத்தபட்டு வேலைக்கான முடிவுகள் எளிதில் பெறப்படுகிறது. இயந்திரம் பழுதுடையும் பொழுது தானாகவே வேறு ஒரு இயந்திரத்திற்கு வேலையை மாற்றி அனுப்பபடுகிறது.

ACKNOWLEDGEMENT

I express my profound gratitude to **Padmabhusan Arutselvar Dr.N.Mahalingam, B.Sc., F.I.E , Chairman**, **Dr.B.K. Krishnaraj Vanavarayar**, **Co-Chairman, Mr. M. Balasubramaniam, M.Com., M.B.A, Correspondent, Mr.Sankar Vanavarayar**, **M.B.A., PGDIEM, Joint Correspondent** and **Dr.S.Ramachandran Ph.D., Principal** for providing the necessary facilities to complete my thesis.

I take this opportunity to thank **Prof.N.Jayapathi M.E.**, Head of the Department, Department of Computer Science and Engineering, for his support and timely motivation. Special thanks to my Project Coordinator **Dr.V.Vanitha Ph.D.**, Senior Associate Professor, Department of Computer Science and Engineering, and project committee members for arranging brain storming project review sessions.

I register my sincere thanks to my guide **Ms.D.Sathya M.E.,** Assistant Professor, Department of Computer Science and Engineering. I am grateful for her support, encouragement and ideas. I would like to convey my honest thanks to all **Teaching** and **Non Teaching Staff** members of the department and my classmates for their support.

I dedicate this project work to my **Parents** for no reasons but feeling from bottom of my heart, without their love this work would not be possible.

S.UMAMAHESWARI

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	**iv**
	LIST OF TABLES	**ix**
	LIST OF FIGURES	**x**
	LIST OF ABBREVIATIONS	**xi**
1.	**INTRODUCTION**	**1**
	1.1 SERVER LOAD BALANCING AND ITS BENEFITS	1
	1.2 OBJECTIVE	5
	1.3 LITERATURE SURVEY	5
	1.3.1 Performance Management for Cluster-Based Web Application	5
	1.3.2 Achieving Self-Scaling in Virtualized Datacenter Management	7
	1.3.3 Precise and Realistic Utility Functions for User Performance Analysis	10
	1.3.4 Dynamic Placement for Clustered Web Application	13
	1.3.5 Learning automated based method for grid computing	15
	1.3.6 Dynamic Resource Allocation of Power Management	17
	1.3.7 Dynamic Resource Allocation to Virtual Containers	21
	1.4 EXISTING SYSTEM	22
	1.4.1 Drawbacks	22
2.	**SYSTEM SPECIFICATION**	**23**
	2.1 HARDWARE SPECIFICATIONS	23
	2.2 SOFTWARE SPECIFICATIONS	23

	2.3 PROPOSED SYSTEM	24
3.	**SYSTEM IMPLEMENTATION**	**25**
	3.1 PROBLEM DEFINITION	25
	3.2 OVERVIEW OF THE PROJECT	25
	3.3 MODULE DESCRIPTION	26
	3.3.1 User Request Extraction	26
	3.3.2 Relative Performance Functionality	26
	3.3.3 Application Program Controller	27
	3.4 DATABASE DESIGN	28
	3.5 SCREEN SHOTS	29
	3.6 RESULT	33
	3.7 CONCLUSION	34
	3.8 FUTURE WORK	34
	APPENDIX	35
	Source Code	35
	REFERENCES	**51**
	LIST OF PUBLICATIONS	**52**

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
1	User Details	28
2	User Request Details	28
3	Bandwidth Details	28

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
1.1	Server Load balancing	2
1.2	Learning Automata	14
2.3	Job Scheduling	24
3.4	Application Program Controller	27
3.5	User Login	29
3.6	User Request	30
3.7	Job Scheduling	30
3.8	Server Allocation	31
3.9	Grid Server Allocation	31
3.10	Grid Response	32
3.11	Grid Server Failure	32
3.12	Job Deadline	33

LIST OF ABBREVIATIONS

APC	Application Program Controller
CJA	Cool Job Allocation
DFS	Dynamic Frequency Scaling
DVS	Dynamic Voltage Scaling
DVFS	Dynamic Voltage and Frequency Scaling
LWPI	Local Workload Placement Index
PC	Program Controller
RPF	Relative Performance functionality

CHAPTER 1

INTRODUCTION

1.1 SERVER LOAD BALANCING AND ITS BENEFITS

Server load balancing is the process of distributing service requests across a group of servers. Server load balancing addresses several requirements that are becoming increasingly important in networks:

• Increased scalability
• High performance
• High availability and disaster recovery

Many content-intensive applications have scaled beyond the point where a single server can provide adequate processing power. Both enterprises and service providers need the flexibility to deploy additional servers quickly and transparently to end-users. The Fig.1.1 shows load balancing within a server farm. End-user requests are sent to a load-balancing device that determines which server is more capable of processing the request and forwards the request to the server. Server load balancing can also distribute workloads to firewalls and redirect requests to proxy servers and caching servers. Server load balancing makes multiple servers appear as a single server and a single virtual service by transparently distributing user requests among the servers. The highest performance is achieved when the processing power of servers is used intelligently.

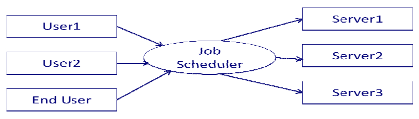


Fig.1.1 Server Load balancing

Advanced server load-balancing products can direct end-user service requests to the servers that are least busy and therefore capable of providing the fastest response time. The load-balancing device should be capable of handling the aggregate traffic of multiple servers. If a server load-balancing device becomes a bottleneck it is no longer a solution, it is just an additional problem. The third benefit of server load balancing is able to improve application availability. If an application or server fails, load balancing can automatically redistribute end-user service requests to other servers within a server farm or two servers in another location. Server load balancing also prevents planned outages for software or hardware maintenance from disrupting service to end-users. Distributed server load-balancing products can also provide disaster recovery services by redirecting service requests to a backup location when a catastrophic failure disables the primary site. This approach provides significant benefits when compared to point products or special purpose appliances:

- Server load balancing is delivered as an overlaid service on the existing network infrastructure. There is no need to redesign the network to accommodate server load balancing.

- True integration provides a simpler and more resilient solution for link, switch, router and load-balancing capabilities.
- Coordinated capabilities for Policy-Based QoS, access policies and system security
- Lower cost of network ownership.

Transactional application and batch jobs are widely used by many organizations to deliver services to their customers and partners. For example, in the financial institution, transactional web workloads are used to trade stocks and query indices, while computationally intensive non interactive workloads are used to analyze portfolios or model stock performance. Due to intrinsic differences between these workloads, they are typically run on separate dedicated hardware, which contribute to resource underutilization and the management complexity. Therefore, organizations demand solutions that permit such workloads to run together on the same hardware, improving resource utilization while continuing to offer performance guarantees. Compute clouds are commonly used by many different users that rely on the existing computing infrastructure to deploy their workloads. The heterogeneous workloads run on the same physical nodes and pose an extraordinary challenge problem. First, the performance goals for different workloads tend to be of different types. For interactive workloads, goals are typically defined in terms of average or percentile response time or throughput over a short time interval, while goals for non interactive workloads concern the performance(e.g., Completion time) of individual jobs.

Second, due to the nature of their goals and short duration of individual request, interactive workloads lend themselves to management at short control cycle, whereas

non interactive workloads typically require calculation of a schedule for an extended period of time. In addition, different types of workload require different control mechanisms for managing. The transactional workloads are managed using flow control, load balancing, and application placement. Interactive workloads are needed for scheduling and resource control. These have been addressed separately. In order to manage resource allocation to a mix of transactional and batch workloads, the system must be able to make placement decisions at short time intervals, so as to respond to changes in transaction workload intensity. While making decisions, the system must be able to look ahead in the queue of jobs and predict the future performance of all jobs. It must able to make trade-offs between the various jobs and the transactional workloads.

The managed system includes a set of heterogeneous server and job scheduler. Long running jobs are submitted to the system via the job scheduler, which unlike traditional schedulers, does not make job execution and placement decisions, the job scheduler only manages dependency jobs and performs resource matchmaking. Once Dependencies are resolved and a set of eligible nodes is determined, jobs are submitted to the application placement controller (APC). APC is the most important component of the system. It provides the decision-making logic that affects placement of both web and non interactive workloads. Its placement optimizer calculates the placement that maximizes the minimum satisfaction across all applications. The Relative Performance Functions (RPF) and Application program Controller (APC) are designed to permit trade-offs between different workloads, and relies on common control mechanisms to manage workloads.

1.2 OBJECTIVE

To reduce the cost of infrastructure and save electrical energy, enterprise data centers consolidate workloads on the same physical hardware. Often, these workloads comprise both transactional and long-running analytic computations. Such consolidation brings new performance management challenges due to the intrinsically different nature of a heterogeneous set of mixed workloads, ranging from scientific simulations to multitier transactional applications. The fact that such different workloads have different natures Imposes the need for new scheduling mechanisms to manage collocated heterogeneous sets of applications, such as running a web application and a batch job on the same physical server, with differentiated performance goals.

1.3 LITERATURE SURVEY

1.3.1 PERFORMANCE MANAGEMENT FOR CLUSTER-BASED WEB SERVICE

E. Arzuaga (2010) proposed that cluster based system it supports the multiple classes of web services traffic and allocate server resources dynamically so to maximize the expected value of a given cluster utility function in the face of fluctuating loads. The cluster utility is a function of the performance delivered to the various classes, and this leads to differentiated service. The system performs the three management tasks: resource allocation, load balancing, and server overload protection. The outer level is a feedback control loop that periodically adjusts the scheduling weights and server allocations of the inner level.

Gateway

Gateways to control the amount of server resources allocated to each traffic class and dynamically changing the amount of resources can control the response time experienced by each traffic class.

Global Resource Manager and Management Console

The global resource manager computes the maximum number of concurrent requests that each server executes on behalf of each gateway and it computes the minimum number of class requests that all servers will execute on the behalf of each gateway. The global resource manager runs periodically and computes the resource allocation parameters every time interval, which define as the i^{th} control horizon. The management console offers a graphical user interface to the management system. Through this interface the service provider can view and override all the configuration parameters.

The Structure of Class Utility Functions

The utility function maps the performance actually experienced by web service requests into a real number Uc. No single way to construct a utility function. When selecting the utility functions, used the following guidelines:

- The value of Uc should be larger when the performance experienced by requests is better than the target and smaller when the performance is worse.

- The value of Uc should increase as the performance experienced by increases and decrease.
- The size and shape of the utility function should be controlled by one or two parameters that can adjust by the platform provider.

1.3.2 ACHIEVING SELF-SCALING IN VIRTUALIZED DATACENTER MANAGEMENT MIDDLEWARE

P. Bodi and C. Sutton (2009) proposed that the increasing popularity of system Virtualization in data centers introduces the need for self-scaling of the management layer to cope with the increasing demands of the management workload. The problem of self scaling in data center management middleware is allowing the management capacity to scale with the management workload. Self-scaling must be fast during workload bursts to avoid task completion delays, and self-scaling must minimize resource usage to avoid resource contention with applications. To address these two challenges, the design of Tide, a self-scaling framework for virtualized data center management. A salient feature of Tide is its fast capacity-provisioning algorithm that supplies just enough capacity for the management middleware. It evaluates the effectiveness of Tide with both synthetic and real world workloads. Results show that the self-scaling capability in Tide can substantially improve the throughput of management tasks during management workload bursts while consuming a reasonable amount of resources.

Self-Scaling and Tide Overview

The intuition behind Tide is to treat the management workload similar to how one would treat an application workload. When a large number of management tasks must be performed, Tide allocates new management instances to handle the additional load, thus preventing a single managed instance (or a small number of statically-partitioned management instances) from becoming a bottleneck in executing the management workload. When the burst subsides, these management instances can reallocate and their underlying physical resources are used for the end-user application workloads, allowing more efficient use of the underlying infrastructure. Such a compute-intensive workload may saturate a single management server. With multiple dynamically-allocated management servers, the management layer can parallelize these computations and minimize the overall latency.

Fast and Efficient Provisioning

Directly estimating the appropriate number of managed instances for a given management workload is difficult partly because the resource consumption of management tasks is hard to predict. Different tasks consume CPU and IO quite differently. Capturing these uncertainties with performance modeling is difficult, and also makes provision system-dependent. One possible solution is to use iterative schemes which repeatedly add new instances into the system until done so does not improve throughput. For instance, a scheme that adds a constant number k of instances each time faces a speed-efficiency dilemma. If k is small, the scheme may take many iterations and a long time to provide sufficient instances. If k is large, it may unnecessarily provision a large amount of instances for small workload bursts, which causes resource contention between management and applications. Compute-intensive workload may saturate a single management server.

Speedup-Guided Provisioning

As N increases, the task throughput increases until the throughput of the system matches the workload. After that, the task throughput levels off even with larger N, because the gain in throughput is limited but the cost of coordinating management instances continues to grow. It refers to the throughput increment after adding one or more instances as throughput speed up, and the state where the throughput levels off as a steady state. In addition, the situation of provisioning is the after steady state of overshooting. Overshooting is clearly undesirable as it wastes resources. Ideally, should stop provisioning new instances when the system enters the steady state.

1.3.3 PRECISE AND REALISTIC UTILITY FUNCTIONS FOR USER-CENTRIC PERFORMANCE ANALYSIS OF SCHEDULERS

B.Chun and D.Culler (2003) proposed that, the utility functions used to represent the value users attach to job completion as a function of turnaround time. Most previous scheduling research used simple synthetic representations of utility, with the simplicity being due to the fact that real user preferences are difficult to obtain, and perhaps concern that arbitrarily complex utility functions could in turn make the scheduling problem intractable. In this work, it advocates a flexible representation of utility functions that can indeed be arbitrarily complex. The genetic algorithm heuristic can improve the global utility by analyzing these functions, and does so tractably. The previous work showed that users indeed have and can articulate complicated utility functions, the result here is relevant.

Modeling Decay Patterns

The first tuple in our utility function is (0, start value), as described above. To generate the remaining tuples, use three different modes of decay in value: expected linear, expected exponential, and step. These represent roughly the three categories of patterns observed in our survey of actual user utility functions. Expected linear represents users whose value decays mostly linearly over time. Expected exponential represents users whose jobs have high initial urgency (a much more precipitous initial decline); however, after some time has passed, additional delay causes a diminishing marginal decay in utility. In both cases, expected refers to the possibility of some “bounce” or random out-of-pattern behavior.

Utility Functions for User-Centric Performance

A utility function is used to represent the value users attach to job completion as a function of turnaround time. Most scheduling research used simple synthetic representations of utility, with the simplicity being due to the fact that real user preferences are difficult to obtain, and perhaps concern that arbitrarily complex utility functions could in turn make the scheduling problem intractable. In this work, advocate a flexible representation of utility functions that can indeed be arbitrarily complex. A genetic algorithm heuristic can improve the global utility by analyzing these functions, and does so tractably. Workload traces with realistic utility functions for the purpose of enabling realistic scheduling simulations.

1.3.4 DYNAMIC PLACEMENT FOR CLUSTERED WEB APPLICATIONS

M.Cardosa and A.Singh (2009) described that middleware clustering technology is capable of allocating resources to web applications through the dynamic application instance placement. Application instance placement as the problem of placing application instances on a given set of server machines to adjust the amount of resources is available to the applications in response to varying resource demands of application clusters. The objective is to maximize the amount of demand that may be satisfied using a configured placement. To limit the disturbance to the system caused by starting and stopping application instances, the placement algorithm attempts to minimize the number of placement changes. It also strives to keep resource utilization balanced across all server machines. Two types of resources are managed, one load-dependent and one load-independent. When putting the chosen placement in effect our controller schedule placement changes in a manner that limits the disruption to the system.

Residual placement

Residual placement is based on the following intuitive rule. Allocated memory at the cost of allocating an application’s CPU demand it is wise to first place an application with the highest memory requirement compared to its CPU demand. This way, if can maximize the chances that this application will be placed on the fewest possible number of nodes, and thus the cost of satisfying its CPU demand will be the lowest. When choosing a node on which to place an application, it is reasonable to first search for a node with a density similar to the density of the application. It is not wise to load applications with high density on a low density server, since would be likely to

reach the processing capacity constraint and leave a lot of memory unused on that server.

Incremental placement

The incremental placement combines the residual placement with the maximum flow computation to solve the placement problem while minimizing the number of placement changes. The number of iterations performed by the incremental algorithms depends on how hard it is to find the satisfying placement. The problem is hard to solve if the total demand of applications compared to the total available capacity is high, or the total memory requirement of applications approaches total memory available on the nodes. The more difficult the problem is to solve, the longer it takes for the algorithm to complete. The upper bound on the number of iterations is the number of assignments in the current placement.

Rebalancing placement

The last phase of the placement algorithm aims at modifying the solution proposed by the incremental algorithm such that a better load balancing across nodes is achieved. This phase begins with old and new placement matrices I* and I and load distribution matrix L. Then try to find another load distribution matrix L+ that satisfies the same demand for all dynamic clusters and that perfectly balances the load assigned to nodes.

- Basic algorithm (BA) - the algorithm as described in the previous section
- Load-reduction algorithm (LRA) - basic algorithm executed with modified input, whenever total application demand exceeds 90% of the total capacity

proportionately reduce the demand of each application so as to bring the total demand down to 90% of the total available capacity

- Multiple-runs an algorithm (MRA) - if placement matrix produced by the basic algorithm does not satisfy the entire demand, execute the basic algorithm one more time using the output of the first execution as a prior - placement matrix.

1.3.5 LEARNING AUTOMATA BASED METHOD FOR GRID COMPUTING RESOURCE VALUATION WITH RESOURCE SUITABILITY CRITERIA

S. David Carrera (2008) described that mechanism based on economic models is an effective approach to solve the problem of grid resources management. Grid resource valuation and allocation is one of the fundamental problems in grid resource management. The essence of this problem is how to allocate and valuation resources for achieving the goal of a highly efficient utilization of resources in response to current resource valuations. Pricing policies are based on the demand from the users and the supply of resources is the main driver in the competitive, economic market model. Present a new method of resource allocation and valuation based on the learning automata algorithms in order to maximize the benefit for both grid providers and grid users.

Learning Automata

Learning Automata are adaptive decision-making devices operating in unknown random environments. A Learning Automaton has a finite set of actions and each action has a certain probability of getting rewarded by the environment of the automaton that has been shown desire response. The aim is to learn to choose the optimal action through repeated interaction with the system. Fig.1.2 shows the learning

algorithm is chosen properly, and then the iterative process of interacting with the environment can made to result in the selection of the optimal action.

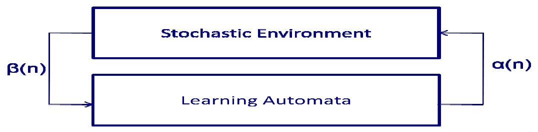


Fig.1.2 Learning Automata

A VSLA is a quintuple < α, β, p, T (α, β, p) >, where α, β, p are an action set with s actions, an environmental response set and the probability set up are containing s probabilities, each being the probability of performing every action in the current internal automaton state, respectively. If the response of the environment takes binary values learning automata model is P-model and if it takes finite output set with more than two elements that take values in the interval [0, 1], such a model is referred to as Q-model, and when the output of the environment is a continuous variable in the interval [0, 1], it is referring to as an S - model.

Grid Resource Allocation Based On Learning Automata

A number of approaches have been proposed which use economic models to address resource usage and incentives in a Grid. Particularly, a well-designed market-based resource allocation mechanism provides incentives for participation by ensuring that all the actors in the system to maximize their utility and do not have incentives to deviate from the designed protocol .It used to provide the new mechanism for grid

resource allocation based on learning automata. After each selection, the LA gets a response from the environment (containing the price of resource and completion time), calculates the complete time, and translates it into a reward R or penalty P for resource number I. The iterative process of interacting with the environment can made to result in the selection of the optimal action.

Pricing Policy

The resource owners may follow the various policies to maximize their profit and resource utilization and the price they charge may vary with time and one user to another user. The pricing can also be driven by demand and supply like in the real market environment. The pricing policy can also be based on sales. In this scale based economic model, pricing is driven by how many users value the service and the highest bidder wins the access to Grid services. Often existing criteria for the grid resource pricing is unrealistic.

Criteria for Selecting Reward and Penalty by Learning Automata

- Run time of transferring application of busiest resources shouldn't have many differences run time with other resource.
- Each work must be assigned to resources that can run it earlier from other resources thus the environment by using the first points will give a penalty to selected action and with second point give reward to the selected action by learning automata.

1.3.6 DYNAMIC RESOURCE ALLOCATION AND POWER MANAGEMENT IN VIRTUALIZED DATA CENTER

D. M David vengerrov and N.Bambos (2007) proposed that investigate optimal resource allocation and power management in virtualized data centers with time-varying workloads and heterogeneous applications. Prior work in this area uses prediction based approaches for resource provisioning. Alternate approach makes use of the queuing information available in the system to make online control decisions. The recently developed technique of Lyapunov Optimization has been to design an online admission control, routing, and resource allocation algorithm for a virtualized data center. This algorithm maximizes a joint utility of the average application throughput and energy costs of the data center. The approach is adaptable to unpredictable changes in the workload and does not require estimation and prediction of its statistics.

CPU Power-Frequency Relationship

Modern CPUs can operate at different speeds at runtime by using techniques such as Dynamic Frequency Scaling (DFS), Dynamic Voltage Scaling (DVS), or a combination Dynamic Voltage and Frequency Scaling (DVFS). These techniques result in a nonlinear power-frequency relationship. This curve was obtained by running a CPU intensive application at different CPU frequencies and utilization levels and measuring the power consumption.

Queuing Dynamics and Control Decisions

Let Ai (t) denotes the number of new request arrives for an application i in slot t. Let Ri(t) be the number of requests out of Ai(t) that are admitted into the Router buffer for an application i by the Admission Controller. Denote this buffer by Wi (t). Assume that any new request that is not admitted by the Admission Controller is declined. This can easily be generalized to the case where arrivals that are not immediately accepted are stored in a buffer for future admission decision. Thus, for all i, t, have: $0 \leq Ri(t) \leq Ai(t)$ (1) Let Rij (t) be the number of requests for application i that are routed from its Router buffer to server j in slot t.

Optimal Stationary, Randomized Policy

It is similar to the general stochastic network utility maximization problem presented in the context of wireless networks with time-varying channels. Specifically, at the beginning of each frame, this policy chooses an active set of servers according to a stationary distributed fashion. Once chosen, other control decisions are likewise taken in a fashion according to stationary distributions.

1.3.7 CAPACITY AND PERFORMANCE OVERHEAD IN DYNAMIC RESOURCE ALLOCATION TO VIRTUAL CONTAINERS

J.Hanson and I.Whalley (2010) proposed that enterprise data centers are shifting towards a utility computing model where many business critical applications share a common pool of infrastructure resources that offer capacity on demand. Management of such a pool requires having a control system that can dynamically allocate resources to applications in real time. Although this is possible by use of virtualization technologies, capacity overhead or actuation delay may occur due to frequent re-scheduling in the virtualization layer. This paper evaluates the overhead of a dynamic allocation scheme in both system capacity and application-level performance relative to static allocation. Because allocation involves re-scheduling in the kernel or the

hypervisor that requires extra computation, it may have the following impact on the system and the hosted applications.

- Capacity overhead**:** Loss of total capacity in the system.
- Performance overhead**:** Loss of capacity in the virtual container, hence degradation of application-level performance.

Two Virtualization Technologies

The two virtualization technologies to create virtual containers are,

Xen: Xen is a hypervisor-based para virtualization technology. It enables multiple Xen domains, i.e., Virtual machines, to be created on a single physical server, each having its own guest OS. The Xen hypervisor provides a software virtualization layer between the guest OS and the underlying hardware. It also provides a credit scheduler that implements weighted fair sharing of the CPU capacity between multiple domains. This scheduler allows each domain to be allocated a certain share of CPU time during a period of time. The scheduler can work in a capped (or non-work-conserving) or non-capped (or work-conserving) mode. In the former, a domain cannot use more than its share of the total CPU time, even if there are idle CPU cycles available, whereas in the latter, a domain can use extra CPU time beyond its share if other domains do not need.

OpenVZ: OpenVZ is a Linux-based OS-level virtualization technology. It can create isolated, secure virtual private servers (VPSs) on a single physical server enabling better server utilization and ensuring a certain degree of isolation between applications. Each VPS performs and executes exactly like a stand-alone server. They can reboot independently and have root access, users, IP addresses, memory, processes, files, applications, system libraries and configuration files. The Open VZ CPU scheduler

implements the FSS strategy. A resource controller running on another computer sends SSH calls to the scheduler periodically to change the resource shares for all the virtual containers. It refers to the period when these changes occur as the control interval. At the same time, each workload also has a sensor that measures its performance (in terms of the amount of work done per unit time) during each control interval. A performance analyzer is also running on another computer to collect both the resource consumption and the performance measurements in order to calculate the possible capacity and performance overhead caused by frequent tuning of the resource shares.

1.3.8 INTEGRATED MANAGEMENT OF APPLICATION PERFORMANCE, POWER AND COOLING IN DATA CENTER

C.B Lee and A.E.Snavely (2007) proposed that Data centers and cooling infrastructures, each of which is typically managed independently. The holistic approach couples the management of IT, power and cooling infrastructures to improve the efficiency of data center operations. These approaches consider application performance management, dynamic workload migration/consolidation, and power and cooling controls to "right-provision" computing, power and cooling resources for a given workload and implement a prototype of this for virtualized environments and conducted experiments in a production data center. The experimental results demonstrate that the integrated solution is practical and can reduce energy consumption of servers by 35% and cooling by 15%, without degrading application performance.

Node Controller

Each node is associated with a node controller. Its role is to maintain the utilization targets for all hosted VMs by dynamically adjusting their resource

allocation. The utilization of a VM as the ratio between its resource consumption and its resource allocation. A node controller consists of a set of utilization controllers (one for each individual VM) and one arbiter. The utilization controller collects the average resource consumption of each VM from sensors and determines the required resource allocation to the VM such that a specified utilization target can achieve. A performance analyzer is also running on another computer to collect both the resource consumption and the performance measurements in order to calculate the possible capacity and performance.

Cooling Management

A traditional CRAC unit in a data center is controlled via the return air temperature. IT equipment inlet temperatures are not measured directly and cannot be assured. The data is used to determine the thermal zones of each CRAC and provide proper cooling to each zone .In this manner, cooling resources can correctly provisioned in real-time according to IT equipment needs.

Local Workload Placement Index

To determine the areas of efficient cooling, use the Local Workload Placement Index (LWPI), first introduced in Cool Job Allocation (CJA), with a modified formulation. LWPI is a measure of how efficiently a location in the data center is cool.

Application Controller

Each application controller controls a single application composed of one or more application components, with each component hosted inside a VM. The policy associated with an application controller includes the response time target, control interval, utilization thresholds, etc. The application controller periodically generates appropriate utilization targets for the corresponding VMs to ensure that the application's performance goals are met. A performance analyzer is also running on another computer to collect both the resource consumption and the performance overhead.

Pod Controller

A pod controller dynamically arranges VM workloads within its pod based on resource requests (e. g., CPU, LAN, SAN and memory needs) of the node controllers associated with servers in the pod. Changes are made via VM live migration. Candidate arrangements are generated and evaluated by a genetic algorithm. The objective function used is designed to consolidate the workload onto the fewest number of nodes possible that can most efficiently cooled and that have the best average utilization. Penalties are assessed for arrangements that include overloaded nodes that require a large number of migrations to achieve, that use nodes to be avoided as per policy, and that use nodes that cannot be efficiently cooled. The terms of the objective function are weighted to achieve a balance between potential performance-disrupting migrations, consolidation and relocation objectives, and the movement of workload to more efficiently cooled areas of the data center.

1.3 EXISTING SYSTEM

In the existing system, clouds are commonly used by many different users that rely on the existing computing infrastructure to deploy their workloads. As a result, heterogeneous workloads run on the same physical nodes and pose extraordinary challenges on a cloud management middleware. Integrated performance management

of mixed workloads is a challenging problem. First, the performance goals for different workloads tend to be of different types. For interactive workloads, goals are typically defined in terms of average or percentile response time or throughput over a short time interval, while goals for no interactive workloads concern the performance (e.g., Completion time) of individual jobs. Second, due to the nature of their goals and the short duration of individual requests, interactive workloads lend themselves to management at short control cycles, whereas non-interactive workloads typically require calculation of a schedule for an extended period of time. In addition, different types of workload require different control mechanisms for managing the transactional workloads are managed using flow control, load balancing, and application placement. Non-interactive workloads need scheduling and resource control.

1.3.2 Drawbacks

- Incoming job nature is not analyzed (i.e. Job nature and workload) before sending to the servers.
- The workloads are not effectively managed.
- Static approach
- Heterogeneous workloads run on the same physical nodes and pose extraordinary challenges on a cloud management middleware.

CHAPTER 2

SYSTEM SPECIFICATION

2.1 HARDWARE REQUIREMENTS

Processor	: Pentium III and above
Clock speed	: 550MHZ
Hard Disk	: 20GB
RAM	: 128MB or above
Cache Memory	: 512KB
Operating System	: Any Windows version
Monitor	: Color Monitor
Keyboard	: 104Keys
Mouse	: 3Buttons

2.2 SOFTWARE REQUIREMENTS

Platform	: Windows 7
Front End	: Java JDK1.5
Back End	: MS SQL server

2.3 PROPOSED SYSTEM

The goal of the technique introduced in this paper is to make placement decisions that involve applications of different nature, more specifically transactional

applications and long-running workloads. The Fig.2.3 shows long-running jobs are submitted to the system via the job scheduler, which, unlike traditional schedulers, does not make job execution and placement decisions. In this system, the job scheduler only manages dependencies between jobs and performs resource matchmaking. Once dependencies are resolved and a set of eligible nodes is determined, jobs are submitted to the application placement controller (APC).

From the point of view of this work, APC is the most important component of the system. It provides the decision-making logic that affects placement of both web and no interactive workloads. Its placement optimizer calculates the placement that maximizes the minimum satisfaction across all applications.

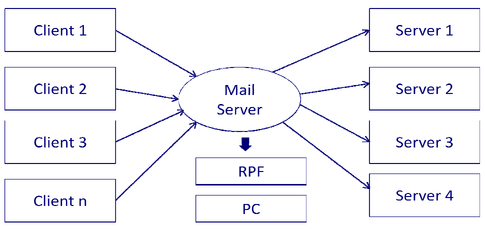


Fig.2.3 Job Scheduling

CHAPTER 3

SYSTEM IMPLEMENTATION

3.1 PROBLEM DEFINITION

Heterogeneous workloads run on the same physical nodes and pose extraordinary challenges on a cloud management middleware. Integrated performance management of mixed workloads is a challenging problem in scheduler.

3.2 OVERVIEW OF THE PROJECT

This system includes a set of heterogeneous server and job scheduler. Long running jobs are submitted to the system via the job scheduler, which unlike traditional schedulers, does not make job execution and placement decisions, the job scheduler only manages dependencies between jobs and performs resource matchmaking. Once dependencies are resolved and a set of eligible nodes is determined, jobs are submitted to the application placement controller (APC). APC is the most important component of the system. It provides the decision-making logic that affects placement of both web and non interactive workloads. Its placement optimizer calculates the placement that maximizes the minimum satisfaction across all applications. The fact that such different workloads have different natures imposes the need for new scheduling mechanisms to manage collocated heterogeneous sets of applications, such as running a web application and a batch job on the same physical server, with differentiated performance goals.

The Relative Performance Functions (RPF) and Application program Controller (APC) are implemented to permit trade-offs between different workloads, and relies on common virtualization control mechanisms to manage workloads. The RPFs define application performance relative to those application goals.

3.3 MODULE DESCRIPTION

The Modules in the proposed system are,
Module 1: User Request Analysis
Module 2: Relative performance functionality
Module 3: Application program controller

3.3.1 User Request Analysis

The user input requests are analyzed by the scheduler before the task is given to the servers. This module helps to avoid the task overloading by analyzing the nature of the users request.

3.3.2 Relative performance functionality

RPF is dealing with the heterogeneous workloads which may differ in their sensitivity to a particular resource allocation. Leverage The flow controller, which comes up with am RPF for each web application this RPF gives a measure of application satisfaction with particular allocation of CPU power given its current workload intensity and performance goal. The Fig.3.4 shows a flow control technique implemented by the flow controller and request router. Each job has an associated performance goal, and when a job completes exactly on schedule, the value of the RPF is zero. Otherwise, the value increase or decrease linearly depending on the distance of

completion time from the goal. Such completion time goals for long running jobs need to be mapped into CPU demand, and vice versa. The required mapping is very hard to obtain for non interactive workloads, because the performance of a given job is not independent of CPU allocation to other jobs. All jobs are not simultaneously run in the system, the completion time of a job that is waiting in the queue for another job to complete before it may be started depends on how quickly the jobs that were started ahead of it completely. Job performance depends on the CPU allocation to another job. In the system implemented heuristics that allow us to estimate CPU requirements for long running jobs for a given value of RPF.

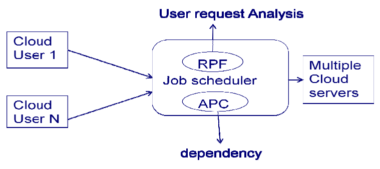


Fig.3.4 Application program Controller

3.3.3 Application program controller

APC is the most important component of the system. The Fig.3.4 shows the decision making logic that affects placement of both web and non interactive workloads. Its placement optimizer calculates the placement that maximizes the minimum satisfaction across all application.APC provide the server load values for identified for job allocation. The different load values are assigned to the server according to the type of the processing file and the nature of the request for server allocation. Depends upon the nature of the request the load values are assigned

dynamically. When the server doesn't respond to the client request and the connection attempt eventually times-out and fails the scheduler switch over the task to another server.

3.4 DATABASE DESIGN

Table design describes the tables maintained in the database and the relationship among the tables. The Table 3.1, Table 3.2 and Table 3.3 show the details about the users, user requests and bandwidth calculations.

Username	Password	IP Address	Port No	Status	Network
Uma	Uma	193.168.10.1	4004	Enable	Network1
Anitha	Anitha	193.168.10.1	4005	Enable	Network1

Table 3.1 User Database

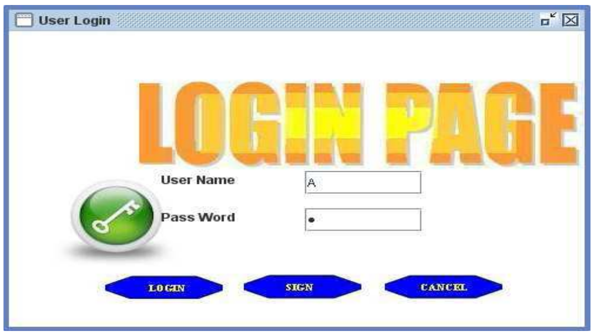
Port No	File Name	IP Address	Value	File Type
4004	Hai.txt	193.168.10.1	3	L
4005	Nature.jpeg	193.168.10.1	3	M
4006	Song.mp3	193.168.10.1	1	H

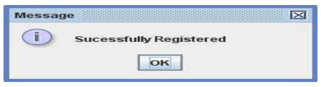
Table 3.2 User Request Details

Server Name	Bandwidth	IP Address	Port No
Grid Server1	34633345	193.168.10.1	4004
Grid Server3	14680916	193.168.10.1	4005
Grid Server3	13774935	193.168.10.1	4006

Table 3.3 Bandwidth Details

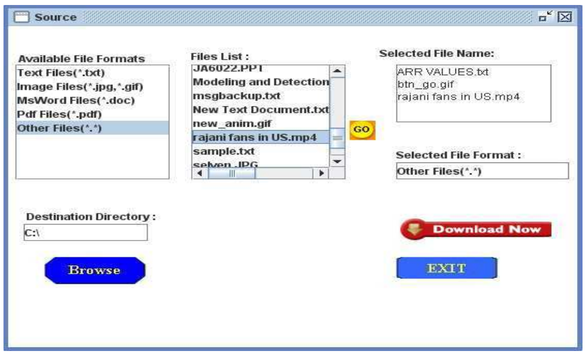
3.5 SCREEN SHOTS





Sucessfully Registered

Fig.3.5 User Login

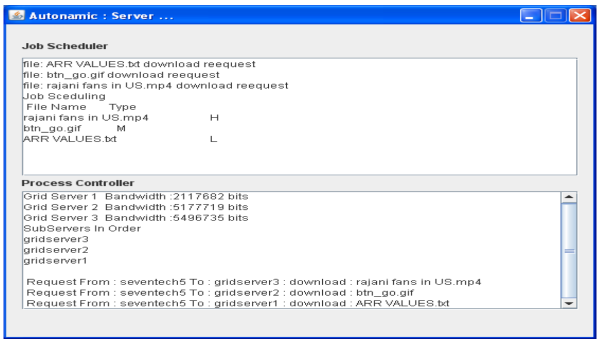


Selected File Name:
ARR VALUES.txt
btn_go.gif
rajani fans in US.mp4

Selected File Format : Other Files(*.*)

Destination Directory : C:\

Fig.3.6 User Request form

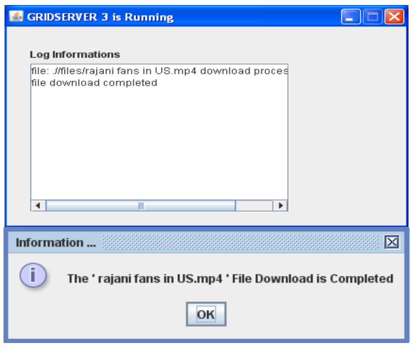


```
file: ARR VALUES.txt download reequest
file: btn_go.gif download reequest
file: rajani fans in US.mp4 download reequest
Job Sceduling
 File Name       Type
rajani fans in US.mp4                 H
btn_go.gif          M
ARR VALUES.txt                        L
```

```
Grid Server 1  Bandwidth :2117682 bits
Grid Server 2  Bandwidth :5177719 bits
Grid Server 3  Bandwidth :5496735 bits
SubServers In Order
gridserver3
gridserver2
gridserver1

 Request From : seventech5 To : gridserver3 : download : rajani fans in US.mp4
 Request From : seventech5 To : gridserver2 : download : btn_go.gif
 Request From : seventech5 To : gridserver1 : download : ARR VALUES.txt
```

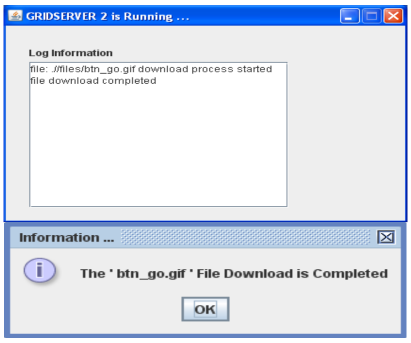
Fig.3.7 Job Scheduling



```
file: ./files/rajani fans in US.mp4 download proces
file download completed
```

The ' rajani fans in US.mp4 ' File Download is Completed

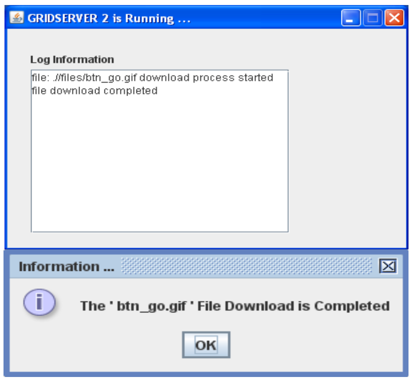
Fig.3.8 Server Allocation



```
file: ./files/btn_go.gif download process started
file download completed
```

The ' btn_go.gif ' File Download is Completed

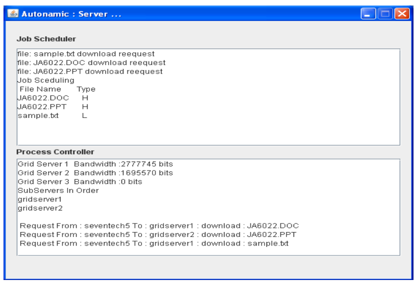
Fig.3.9 Grid Server Allocation



```
file: ./files/btn_go.gif download process started
file download completed
```

The ' btn_go.gif ' File Download is Completed

Fig.3.10 Grid Server Response



```
file: sample.txt download reequest
file: JA6022.DOC download reequest
file: JA6022.PPT download reequest
Job Sceduling
 File Name      Type
JA6022.DOC     H
JA6022.PPT     H
sample.txt       L
```

```
Grid Server 1  Bandwidth :2777745 bits
Grid Server 2  Bandwidth :1695570 bits
Grid Server 3  Bandwidth :0 bits
SubServers In Order
gridserver1
gridserver2

 Request From : seventech5 To : gridserver1 : download : JA6022.DOC
 Request From : seventech5 To : gridserver2 : download : JA6022.PPT
 Request From : seventech5 To : gridserver1 : download : sample.txt
```

Fig.3.11 Grid Server Failure

3.6 RESULT

The comparative of APC with simple, effective, and well-known scheduling algorithms, earliest Deadline First (EDF) and First-Come, First-Served (FCFS).EDF is a preemptive scheduling algorithm, FCFS does not preempt jobs. In both cases, a first-fit strategy was followed to place the jobs. Use eight different interval times, ranging from 50 to 400 s, and continue to submit jobs until 800 have completed. The experiment is repeated for the three algorithms: APC, EDF, and FCFS. Here concentrate on the results for interarrival times of 200 and 50s due to space limitations. Fig.3.12 shows the percentage of jobs that met their completion time goal. There is no significant difference between the algorithms when interarrival times are greater than 100 s—this is expected, as the system is under loaded in this configuration. However, with an interval period of 100 s or less, FCFS starts struggling to make even 50 percent of the jobs meet their goals. EDF and APC have a significantly higher, and comparable, ratio of jobs that met their goals. At a 50 s interval time, the goal satisfaction rate for FCFS has dropped to 40 percent, and the goal satisfaction rate is actually higher for EDF than for APC.

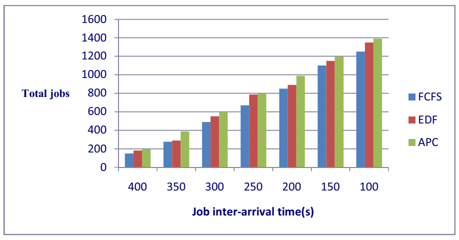


Fig.3.12 Percentage of jobs met the deadline

3.7 CONCLUSION

The system has been implemented and integrated with a commercial application server middleware, which provides the support for executing placement decisions. This system is driven by high-level application goals and takes into account the application satisfaction with how well the goals are met. Have demonstrated, both using a real-system and Java, this approach improves satisfaction fairness across applications compared to existing state of the art solutions. The system introduces several novel features. First, it allows heterogeneous workloads to be collocated on any server machine, thus reducing the granularity of resource allocation. Second, the approach uses high-level performance goals (as opposed to lower-level resource requirements) to drive resource allocation. Third, technique exploits a range of new automation mechanisms that will also benefit a system with a homogeneous, particularly non-interactive workload by allowing more effective scheduling of jobs.

3.8 FUTURE ENHANCEMENTS

The technique aims to permit SLA-based placement of transactions and batch workloads, also wanted to evaluate its efficacy in the management of batch workloads alone, as a scheduling mechanism. In this context, approach to scheduling differs from traditional techniques in that it concentrates on achieving fair performance levels among all jobs rather than on optimizing the overall system throughput. The technique is presented as an alternative to traditional batch scheduling mechanisms for enterprise data centers and cloud computing infrastructures in which resource utilization is a critical goal even in the presence of heterogeneous workloads.

APPENDIX 1

SOURCE CODE

Job Scheduling

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.sql.*;
import java.lang.*;
import java.util.Vector;
import java.awt.Toolkit;
import java.util.Timer;
import java.util.TimerTask;
import java.util.StringTokenizer;
import java.util.concurrent.*;
class server extends JFrame implements rouconfig
{
String systemName;
```

```
String cuurentip="";
 int change;
Vector system=new Vector();
int curport,thtime,restime;
 int sto,ttime=0;
String mesg,process,file,sysname,s,jport;
 ResultSet rs;
 Connection con;
 Statement st;
 DefaultListModel model1;
 private JLabel jLabel1;
 private JLabel jLabel3;
 private JLabel jLabel3;
 private JLabel jLabel4;
 private JTextArea jList1;
 private JScrollPane jScrollPane1;
 private JTextArea jTextArea1;
 private JScrollPane jScrollPane3;
 private JButton jButton1;
 private JButton jButton3;
 private JButton jButton3;
 private JButton jButton4;
 private JPanel contentPane;
Vector data=new Vector();
ArrayList a1=new ArrayList();
ArrayList a3=new ArrayList();
ArrayList allname=new ArrayList();
String logsys,log,sname,all,desname,smesg;
Thread t1;
String source,destination,state,smsg,srcname,msgtime;
FileOutputStream output,op1;
String fd;
boolean sta=true,stak;
int end;
Socket  cs,cs1,cs3;
ObjectInputStream in,in1,in3;
ObjectOutputStream ois1,ois3,ois3;
String res1;
 int  staa=0,che;
String hh;
Vector v1,v3;
String name[]=new String[100];
String allsta;
int portno;
```

```
int numWarningBeeps=1 ;
boolean sss;
Vector p3=new Vector();
Vector text=new Vector();
Vector allfile=new Vector();
ServerSocket ss=new ServerSocket(1111);
server()throws Exception
{
JLabel bg;
bg = new JLabel(new ImageIcon("predictionserver.jpg"));
portno=6666;
v1=new Vector();
JLabel jLabel7=new JLabel();
allsta="allow";
jLabel1 = new JLabel();
jLabel3 = new JLabel();
jLabel3=new JLabel();
jLabel4 = new JLabel();
model1=new DefaultListModel();
jList1 = new JTextArea();
jScrollPane1 = new JScrollPane();
jTextArea1 = new JTextArea();
```

```
jScrollPane3 = new JScrollPane();
jButton1 = new JButton();
jButton3 = new JButton();
jButton3 = new JButton();
jButton4 = new JButton();
contentPane = (JPanel)this.getContentPane();
setResizable(false);
jLabel1.setText("Job Scheduler");
jLabel3.setText("Process Controller");
jScrollPane1.setViewportView(jList1);
jScrollPane3.setViewportView(jTextArea1);
addWindowListener(new WindowAdapter()
{
public void windowClosing(WindowEvent e)
{
System.exit(0);
}
}
String rew="";
contentPane.setLayout(null);
addComponent(contentPane, jLabel7, 50,30,130,130);
addComponent(contentPane, jLabel1, 15,35,150,15);
```

```
addComponent(contentPane, jLabel3, 15,330,150,15);
addComponent(contentPane, jScrollPane1, 15,50,535,177);
addComponent(contentPane, jScrollPane3, 15,350,535,177);
addComponent(contentPane, bg, 50,30,130,130);
MouseListener mouseListener = new MouseAdapter() {
public void mouseClicked(MouseEvent mouseEvent) {
JList jList1 = (JList) mouseEvent.getSource();
 if (mouseEvent.getClickCount() == 3) {
 int index = jList1.locationToIndex(mouseEvent.getPoint());
 if (index >= 0) {
Object o = jList1.getModel().getElementAt(index);
String tatext=jTextArea1.getText();
}
}
}
 };
jList1.addMouseListener(mouseListener);
jButton3.setEnabled(false);
this.setTitle("Autonamic : Server ...");
this.setLocation(new Point(100, 100));
this.setSize(new Dimension(560, 500));
this.setVisible(true);
```

```
}
private void addComponent(Container container,Component c,int x,int y,int width,int height)
{
c.setBounds(x,y,width,height);
container.add(c);
}
private void jList1_valueChanged(ListSelectionEvent e)
{
}
  public void msg()
{
try
{
allfile.removeAllElements();
Socket s3=new Socket(systemName,3333);
ObjectOutputStream   oos1=new ObjectOutputStream(s3.getOutputStream());
oos1.writeObject("fileList");
in=new ObjectInputStream(s3.getInputStream());
allfile=(Vector)in.readObject();
System.out.println(" File List Receive from server 1 \n\n\n-----\n\n"+allfile+"\n\n--------\n\n");
//jList1.setListData(allfile);
System.out.println( "\nFiles in this directory are:" );
```

```
System.out.println( allfile );
}
catch (Exception e34)
{
}
curport=3333;
int oldrestime=0;
int newrestime=0;
while(true)
{
System.out.println("Server is listening........");
try
{
cs=ss.accept();
in=new ObjectInputStream(cs.getInputStream());
String resu=(String)in.readObject();
System.out.println(" First Client Request  :"+resu);
class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con = DriverManager.getConnection("jdbc:odbc:Driver={SQL
Server};Database=server;Server=.;UID=sa");
st=con.createStatement();
StringTokenizer tokens = new StringTokenizer(resu,"$");
//String tokensarray[]=resu.split("$");
```

```
while(tokens.hasMoreTokens())
{
//System.out.println(tokens.nextToken());
process=tokens.nextToken();
file=tokens.nextToken();
sysname=tokens.nextToken();
jport=tokens.nextToken();
// s="insert into work values('"++"','"++"',)";
 //rs=st.executeUpdate(s);
}
String filehl[]=file.split("&");
if(process.equals("select"))
{
 s="delete from filetable";
st.executeUpdate(s);
//jTextArea1.setText(jTextArea1.getText()+"\n"+process +" : "+ file+ " : "+process);
change=0;
p3.removeAllElements();
text.removeAllElements();
for(int count=0;count<allfile.size();count++)
{
String che=(String)allfile.elementAt(count);
```

```
p3.add(che);
/System.out.println(count+" == "+che+"\n\n");
if(che.endsWith(file))
{
text.add(che);
}
if(file.equals("(*.*"))
{
if(!che.endsWith("*.txt") &&!che.endsWith("*.jpg") && !che.endsWith("*.gif") &&
!che.endsWith("*.pdf") )
{
text.add(che);
}
}
}
ois1=new ObjectOutputStream(cs.getOutputStream());
if(change==0)
{
ois1.writeObject(text);
}
else
{
ois1.writeObject(p3);
```

```
}
//System.out.println(" Inside the Text file "+text);
}
else
{
jList1.setText("");
jTextArea1.setText("");
String type="",zzz="";
for(int i=0;i<filehl.length;i++)
{
jList1.append("file: "+filehl[i]+ " download reequest \n");
String filenz=filehl[i];
if(filenz.endsWith(".txt") || filenz.endsWith(".doc"))
{
type="3";
zzz="L";
}
else if(filenz.endsWith(".gif") || filenz.endsWith(".jpg") || filenz.endsWith(".jpeg"))
{
type="3";
zzz="M";
}
```

```
else
{
type="1";
zzz="H";
}
 s="insert into filetable values('"+jport+"','"+filehl[i]+"','"+sysname+"','"+type+"','"+zzz+"')";
 st.executeUpdate(s);
}
String fileh[]=new String[filehl.length];
String filez[]=new String[filehl.length];
int j=0;
s="select * from filetable order by type";
rs=st.executeQuery(s);
while(rs.next())
{
fileh[j]=rs.getString(3);
filez[j]=rs.getString(5);
 j++;
}
jList1.append("Job Sceduling \n File Name       Type\n");
For(int i=0;i<fileh.length;i++)
{
```

```
jList1.append(fileh[i]+"\t"+filez[i]+"\n");
}
s="delete from work";
st.executeUpdate(s);
long start=0;
long end=0;
long resdelay=0;
int newrestime1=0;
int newrestime3=0;
int newrestime3=0;
int bandwidth1=0;
int bandwidth3=0;
int bandwidth3=0;
Socket s3;
String req="";
ObjectOutputStream oos1;
try
{
start=System.nanoTime();
s3=new Socket(serverip1,serverport1);
oos1=new ObjectOutputStream(s3.getOutputStream());
oos1.writeObject("hai33");
```

```
in=new ObjectInputStream(s3.getInputStream());
req=(String)in.readObject();
System.out.println(" Receive from server 1 "+req);
end=System.nanoTime();
resdelay=end-start;
newrestime1=(int)resdelay();
//System.out.println("\n\nnewrestime1: "+newrestime1);
if(newrestime1==0)
newrestime1=1;
bandwidth1=(33*1000)/newrestime1;
System.out.println("bandwidth1:"+ newrestime1);
}
catch(Exception ec)
{
newrestime1=0;
}
jTextArea1.append("Grid Server 1  Bandwidth :"+ newrestime1+" bits \n");
s="insert into work values('gridserver1','"+newrestime1+"','"+serverip1+"','"+serverport1+"')";
st.executeUpdate(s);
try
{
start=System.nanoTime();
```

```
s3=new Socket(serverip3,serverport3);

oos1=new ObjectOutputStream(s3.getOutputStream());

oos1.writeObject("hai33");

in=new ObjectInputStream(s3.getInputStream());

req=(String)in.readObject();

System.out.println(" Receive from server 3 "+req);

end=System.nanoTime();

resdelay=end-start;

newrestime3=(int)resdelay;

//System.out.println("\n\nnewrestime3: "+newrestime3);

if(newrestime3==0)

newrestime3=1;

bandwidth3=(33*1000)/newrestime3;

System.out.println("bandwidth3:"+ newrestime3);

}

catch (Exception el)

{

newrestime3=0;

}

jTextArea1.append("Grid Server 3  Bandwidth :"+ newrestime3+" bits \n");

s="insert into work values('gridserver3','"+newrestime3+"','"+serverip3+"','"+serverport3+"')";

st.executeUpdate(s);
```

```
try

{

start=System.nanoTime();

s3=new Socket(serverip3,serverport3);

oos1=new ObjectOutputStream(s3.getOutputStream());

oos1.writeObject("hai33");

in=new ObjectInputStream(s3.getInputStream());

req=(String)in.readObject();

System.out.println(" Receive from server 3 "+req);

end=System.nanoTime();

resdelay=end-start;

newrestime3=(int)resdelay;

/System.out.println("\n\nnewrestime3: "+newrestime3);

 if(newrestime3==0)

 newrestime3=1;

bandwidth3=(33*1000)/newrestime3;

System.out.println("bandwidth3:"+ newrestime3);

}

catch(Exception ez)

{

newrestime3=0;

}
```

```
jTextArea1.append("Grid Server 3  Bandwidth :"+ newrestime3+" bits \n");

S="insert into work
values('gridserver3','"+newrestime3+"','"+serverip3+"','"+serverport3+"')";

st.executeUpdate(s);

String portid[]=new String[3];

String cip[]=new String[3];

int cport[]=new int[3];

int il=0;

s="select * from work order by bandwidth desc";

rs=st.executeQuery(s);

}
```

REFERENCES

1. E.Arzuaga and D.R.Kaeli, “Performance Management For Cluster-Based Web Service”,Proc. First Joint WOSP/SIPEW Int’l Conf. Performance Eng. (WOSP/SIPEW ’10), pp. 335-343, 2010.

2. P.Bodı´k,R.Griffith,C.Sutton,A.Fox,M.I.Jordan,and D.A.Patterson, “Achieving Self-Scaling In Virtualized Datacenter Management Middleware,” Proc. Conf. Hot Topics in Cloud Computing (HotCloud ’09), 2009.

3. B.N.Chun and D.E.Culler, “Precise and Realistic Utility Functions For User-Centric Performance Analysis Of Schedulers,” Proc. IEEE/ACM Second Int’l Symp. Cluster Computing and the Grid, p. 30, 2003.

4. M.Cardosa,M.R.Korupolu,and A.Singh, “Dynamic Placement For Clustered Web Applications,” Proc. IFIP/IEEE Int’l Symp. Integrated Network Management (IM ’09),pp. 337-334, 2009.

5. S.David Carrera,and Sun Microsystems, “Learning Automata Based Method For Grid Computing Resource Valuation With Resource Suitability Criteria”, Technicalreport,http://www.sun.com/software/resourcemgr/wp-mixed/wp-mixed.pdf, Jan 2009.

6. D.M. David Vengerov,L.Mastroleon, and N. Bambos, “Dynamic Resource Allocation And Power Management In Virtualized Data Center,” Technical Report TR-3007-164, Sun Microsystems, Apr.2007.

7. J.Hanson,I. Whalley, M.Steinder, and J.Kephart, “Capacity And Performance Overhead In Dynamic Resource Allocation To Virtual Containers ,” Proc.IEEE Network Operations and Management Symp. (NOMS), pp. 543-550, Apr. 2010.

8. C.B.Lee and A.E.Snavely, “Integrated Management Of Application Performance, Power And Cooling In Data Center,” HPDC ’07:Proc. 16th Int’l Symp. High Performance Distributed Computing,pp. 107-116, 2007.

9. J.G.Park,J.-M.Kim,H.Choi,and Y.-C. Woo, “Virtual Machine Migration in Self-Managing Virtualized Server Environments,”Proc. 11th Int’l Conf. Advanced Comm. Technology (ICACT), pp. 3077-3083, 2009.

LIST OF PUBLICATIONS

1. S.Umamaheswari and D. Sathya, "Autonomic Placement of Heterogeneous Workloads", National Conference on Innovations in Information Technology, Bannari Amman Institute of Technology, Feb.2013.

2. S.Umamaheswari and D.Sathya, "Autonomous Assignment of Mixed Workloads", International Conference on Advanced Computing, Machines and Embedded Technologies, J.K.K.N College of Engineering and Technology, Feb. 2013.