P-3298

AN EFFICIENT SCHEDULING DATA STRUCTURE FOR GRID ADVANCE RESERVATION

By

R.VINOTH KUMAR

Reg. No: 0820108024

of

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna university, Coimbatore)

COIMBATORE – 641 006

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements
for the award of the degree
of

MASTER OF ENGINEERING
IN

COMPUTER SCIENCE AND ENGINEERING

MAY 2010

BONAFIDE CERTIFICATE

Certified that this project report titled "**An efficient scheduling data structure for Grid Advance Reservation**" is the bonafide work of **Mr.R.Vinothkumar (0820108024)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report of dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

GUIDE

(Ms.S.RAJINI)

HEAD OF THE DEPARTMENT

(Ms.P.DEVAKI)

The candidate with **University Register No. 0820108024** was examined by us in Project Viva-Voce examination held on 18/5/10

18/5/10

Internal Examiner

External Examiner

VIVEKANANDHA

INSTITUTE OF ENGINEERING AND TECHNOLOGY FOR WOMEN

ELAYAMPALAYAM, TIRUCHENGODE.

Department of Computer Applications

NATIONAL CONFERENCE

ON

" EMERGING TECHNOLOGIES IN ADVANCED COMPUTING AND COMMUNICATION "

13th March, 2010

CERTIFICATE

This is to Certify that Mr. / ~~Ms.~~ / ~~Dr.~~ R. VINOTHKUMAR of II ME (CSE), Kumaraguru College of Technology has participated in the " National Conference on ETACC '10 ", organized by " VIVACIOUS " the professional association of Computer Applications on 13th March 2010 and presented a paper on Scheduling With Advanced Reservation.

A. Ahamed Nahaal
HOD & Organizing Secretary

Principal

Chairman & Secretary

ABSTRACT

In Grids, users may require assurance for completing their jobs on shared resources. Such guarantees can only be provided by reserving resources in advance. However, if many reservation requests arrive at a resource simultaneously, the overhead of providing such service due to adding, deleting, and searching, will be significant. An efficient data structure for managing these reservations plays an important role in order to minimize the time required for searching available resources, adding, and deleting reservations. In this report, a presentation about the new approaches to advance reservation in order to deal with the limitations of the existing data structures, such as Segment Tree and Calendar Queue in similar problems has been done. An efficient data structure using Grid advanced reservation Queue (GarQ), which is a new data structure that improves some weaknesses of the aforementioned data structures is proposed to be implemented and tested.

ஆய்வுச் சுருக்கம்

இந்த ஆய்வு உணர்த்துவது என்னென்றால், இந்த கட்டடத்தில் பல பயனாளர்கள் அவர்களின் வேலையை வளத்தின் மூலமாக பகிர்ந்துக்கொள்கிறார்கள்.ஆனால் இங்கு வளம் உத்திரவாதம் நிச்சியமாக முன்பதிவு செய்வதன் மூலமாக வழங்கப்படுகிறது.எப்படி இருந்தாலும் முன்பதிவு ஒரே நேரத்தில் செய்யும் போது சேர்த்தல்,அழித்தல்,தேடுதல் சற்று கடினம்.இப்பொழுது ஆற்றல் மிக்க தரவு அமைப்பு மூலம் முன்பதிவு எளிதில் செய்ய இயலும் மற்றும் நேரம் குறையும். இங்கு உள்ள பழையை தரவு அமைப்பு நாள்காட்டி வரிசை மற்றும் பிரித்தல் மரம் சற்று குறைகள் உள்ளன. கட்டம் முன் கூட்டியே முன்பதிவு மூலம் நாம் எளிதில் வேலையையும்,வளத்தையும் பகிர்ந்து கொள்கிறோம். இந்த ஆய்வின் மூலம் நாம் எளிதில் முன்பதிவின் மூலம் வேலையை வளத்தின் உதவியுடன் செய்துக்கொள்கிறோம்.

ACKNOWLEDGEMENT

I express my profound gratitude to our Chairman **Padmabhusan Arutselver Dr. N. Mahalingam B.Sc, F.I.E** for giving this great opportunity to pursue this course.

I would like to begin by thanking to Dr.S. **Ramachandran**, *Principal* for providing the necessary facilities to complete my thesis.

I take this opportunity to thank **Dr. S.Thangasamy,** R & D Dean , for his precious suggestions.

I register my hearty appreciation to **Ms. S.Rajini, Assistant professor**, my thesis advisor. I thank her for support, encouragement and ideas.

I thank all project committee members for their comments and advice during the reviews. Special thanks to **Ms.V.Vanitha, M.E.,** *Assistant professor*, Department of Computer science and Engineering, for arranging the brain storming project review sessions.

I would like to convey my honest thanks to all **Teaching** staff members and **Non Teaching** staffs of the department for their support. I would like to thank all my classmates for their help.

I dedicate this project work to my **parents and friends.**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

GARQ	Grid Advanced Resource Queue
QOS	Quality of Services
AR	Advance Reservation
GIS	Grid Information System
FES	Future Event Set
CNs	Computing Nodes
RV	Reservation
totAR	Total number of Advance Reservation
leftS	Left SubTree
rightS	Right SubTree
ts	Starting Time
te	Ending Time
numCN	Number of Computing Node

1. INTRODUCTION

1.1 OVERVIEW OF GRID COMPUTING

Grid computing is a form of distributed computing that involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations. Grid technologies promise to change the way organizations tackle complex computational problems.

Grid computing enables the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities.

Grid Computing is based on an open set of standards and protocols – e.g., Open Grid Services Architecture (OGSA) – that enable communication across heterogeneous and geographically dispersed environment.

Computational grid is a collection of distributed, possibly heterogeneous resources which can be used as an ensemble to execute large scale applications.

Grid applications include

- Distributed supercomputing
    - Distributed supercomputing application couple multiple computational resources such as supercomputers and/or workstation

- High- Throughput Applications
    - Grid is used to schedule large numbers of independent or loosely coupled tasks with the goal of putting unused cycles to work. High-throughput applications include RSA key cracking detection of extra-terrestrial communication.

- Data-Intensive Applications
    - Focus is on synthesizing new information from large amounts of physically distributed data. Examples include NILE (distributed system for high energy physics experiments using data from CLEO), SAR/SRB application, and digital library application.

1.1.1 GRID CONSTRUCTION

There are three main issues that characterize computational grids:

Heterogeneity:

A grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances.

Scalability:

A grid might grow from few resources to millions.

Dynamicity or Adaptability:

Since many resources are presented in a grid, the probability of some resource failing is naturally high.

The steps necessary to realize a computational grid, include

- The integration of individual software and hardware components into a combined network resource.

- The implementation of middleware to a transparent view of the resources available.
- The development of tools that allows management and control of grid applications and infrastructure

1.1.2 GRID COMPONENTS

Grid Fabric:

It comprises all the resources geographically distributed and accessible from anywhere on the internet. They could be computers, cluster, storage devices, databases, and special scientific instruments such as a radio telescope.

Grid Middleman:

It offers core services such as remote process management, co-allocation of resources, storage access, information, security, authentication, and quality of services (Q0S) such as resource reservation and trading.

Grid Development Environment and Tools:

These offer high-level services that allow programmers to develop applications and brokers that act as user agents that act as user agents that can manage or schedule computations across global resources.

Grid Application and Portals:

They are developed using grid- enabled languages such as HPC++, and message system such as MPI. Application, such as parameter simulations and grand- challenge server they are accessing on a UNIX or NT platform.

1.1.3 TYPES OF GRIDS

National- Grids

They provide a strategic "computing reserve" and will allow substantial computing resources to be applied to large problems in times of crisis, such as

to plan responses to a major environmental disaster, earthquake, or terrorist attack. Further one word more, such a grid will act as a "national collaborator", supporting collaborative investigation of complex scientific and engineering problems, such as global climate change, space station design ,and environmental cleanup.

Private-Grids

Private grids are useful in many institutions (hospital, corporation, small firms, etc). They are characterized by a relatively small scale, central management and common purpose and, in most cases, they will probably need to integrate low cost commodity technology.

Project-Grids

Project grids are be created to meet the needs of a variety of multi institutional research group and multi company "VIRTUAL TEAMS" to pursue short or medium term projects (scientific collaboration, engineering projects). A project grid will typically be build ad hoc from shared resources for a limited time, and it focused on a specific goal.

Goodwill – Grids

Goodwill grids are for any one owning a computer at home who wants to donate some computer capacity to a good cause.

Peer-Peer-Grids

Peer –to-peer technology depends on people sharing data between their computers. Peer –to-peer suggests that there is no central control.

Consumer Grids

In a consumer grid, resources are shared on a commercial basis, rather than on the basis of goodwill or mutual self-interest. A big issue in such grids

will be "RESOURCES MARKETING" a user has to find the resources needed to solve this particular problem, and the supplier must make potential users aware of the resources he has to offer.

1.2 ADVANCE RESERVATION

Advance reservation is the process of requesting resources for use at specific times in the future. Common resources that can be reserved are compute nodes and network bandwidth.

In order to reserve the available resources in such AR systems, a user must first submit a request by specifying a series of parameters such as number of compute nodes needed, and start time and duration of his/her jobs. Then, the AR system checks for the feasibility of this requests. If there are no available nodes for this requested time period, the request are rejected. Consequently, the user may resubmit a new request with a different start time and/or duration until available nodes can be found.

The choice of efficient data structure can significantly minimize the time complexity needed to search available compute nodes, add new requests, and delete existing reservations. Moreover, a well-designed data structure provides the flexibility and easiness in implementing various algorithms.

Types of request

- Designate request
    - A request for specific physical resources
- Abstract request
    - A request for resources with specified capabilities (attributes). Mapping to physical resources required

1.3 SCHEDULING SYSTEM

Advance Reservation (AR) in a scheduling system solves the above problem by allowing users to gain simultaneous and concurrent access to adequate resources for the execution of such applications. Currently, several Grid systems are able to provide AR functionalities.

Scheduling and provisioning

- Scheduling
    - Allocating using right of resources to a request
    - If the request is an abstract request, mapping between the request and physical resources is also determined
- Provisioning
    - Activate allocated resources and make them usable by the requester.
    - May be automatic / may be kicked by additional operation by the requester (signaling)

Types of scheduling

- On-demand
    - If the resources are available when a request is issued, the resources are allocated to the requester.
    - If the request is an abstract request, mapping of physical resources is done.
    - If the resources are not available, the request is simply denied.
    - Usually, the requester can use the allocated resources as long as it wants. The end time of the provisioning is not determined at the time of provisioning.

- Batch
  - Widely used in computing systems such as supercomputing centers.
  - Requests are queued in a queue, and the request at the head of the queue is scheduled when required resources become available If the request is an abstract request, mapping of physical resources is done at this moment.
  - May have multiple prioritized queues.

- Advance reservation
  - A request is processed by a scheduler, and the scheduler finds a period when the requested resources are available for the requested duration
  - The resources are reserved during the reservation period, and when the reservation period begins, the resources are allocated to the requester.
  - A reservation database is maintained by the scheduler or resource managers, and referred and updated by the scheduler.
  - Scheduling is done when a request is issued. In addition, re-scheduling may be done when availability of resources is changed.

- Instant reservation
  - If an advance reservation request requests immediate allocation of resources, the request is called "instant reservation".
  - Different from the pure "on-demand" the end time of a provisioning is determined at the time of scheduling.

2. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

The previous studies are primarily focused on finding out the search time of the aforementioned data structures. However, these studies do not explicitly consider add and delete operations for adding new requests and deleting existing reservations respectively, for these data structures. This is because, for reserving network bandwidth, each tree node and index in Segment Tree and Array respectively, only stores information regarding the allocated reserved bandwidth. Hence, the performance of addition and deletion can be neglected. In contrast, a data structure needs to keep additional information for reserving compute nodes in Grid systems, such as user's jobs for executing on the reserved nodes, and their status for monitoring purposes. This operation helps users who requests got rejected in negotiating a suitable reservation time.

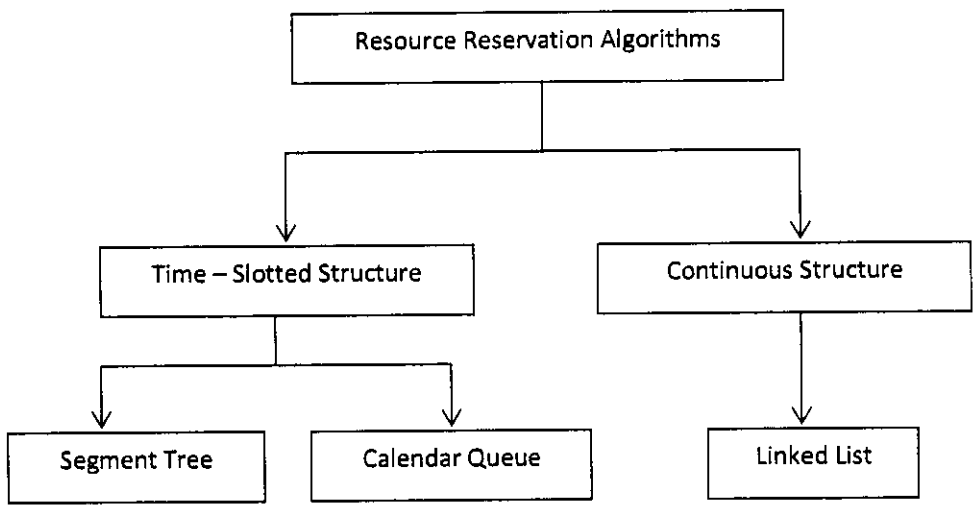
They primarily focused on finding out the search time of the data structures.

1. Segment tree
2. Calendar queue
3. Linked list

It is not explicitly consider add and delete operations for adding new requests and deleting existing reservations respectively.

These data structures for reserving network bandwidth, each tree node and index in Segment Tree and Array respectively, only stores information regarding the allocated reserved bandwidth. Hence, the performance of addition and deletion can be neglected.

It is not consider an interval search operation, where the data structure finds an alternative time for a rejected request. This operation helps users who requests got rejected in negotiating a suitable reservation time. Therefore, the performance of this operation also needs to be considered when choosing the appropriate data structure.



2.1.1 TREE BASED DATA STRUCTURE

Tree-based data structure is commonly used for admission control in network bandwidth reservation. Each tree node contains a time interval and the amount of reserved bandwidth in its sub tree. Therefore, a leaf node has the smallest time interval compared to its ancestor nodes. Hence, the number of bandwidth required for a single reservation is stored into one or more fitting nodes.

In general, a tree-based structure has a time complexity of O(log n) for searching the available bandwidth, where n is the number of tree nodes.

2.1.1.1 SEGMENT TREE

For processing new requests and searching larger time intervals. The study was conducted to measure the admission speed of a bandwidth broker using each structure in a multilink admission control environment.

Segment Tree is a binary tree where each node represents a semi-open time interval (X; Y]. The left sibling of the node represents the interval (X; X+Y/2] and the right sibling represents the interval (X+Y/2 ; Y] Each node has also the following information:

1) rv: the number of reserved CNs over the entire interval. When a reservation which spans the entire interval (X; Y] is added, rv is increased by the number of CNs required by this reservation. No further descent into the child nodes is needed.

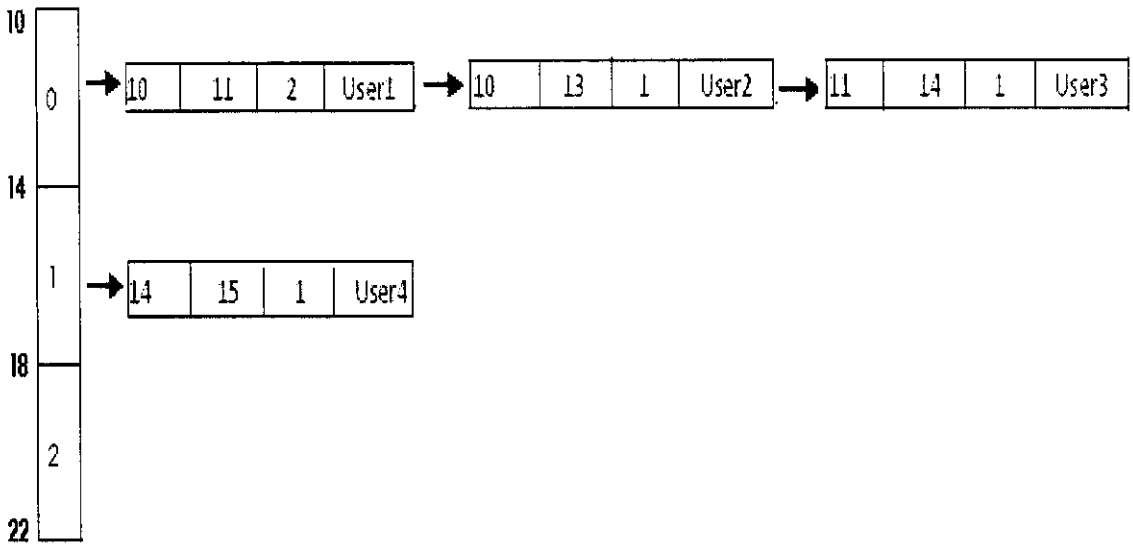
2) mv: the maximum number of reserved CNs in the child nodes. In the leaf nodes, the mv value is 0. The total number of reserved CNs in the interval of a leaf node is the sum of all rv of nodes on the path from the root node to the leaf node.

2.1.1.2 CALENDAR QUEUE

Calendar Queue (CalendarQ) was introduced by Brown, as a priority queue for future event set problems in discrete event simulation.

It is modeled after a desk calendar, where each day or page contains sorted events to be scheduled on that period of time. Hence, CalendarQ is represented as one or more pages r "bucket" with a fixed time interval or width sigma. Then, each bucket contains a sorted linked list storing future events.

If a reservation requires more than sigma, this reservation will also be duplicated into the next buckets. This approach makes the search operation easier since it only searches for a list inside each bucket.



Data structure has buckets with a fixed sigma, which represents the smallest slot duration, as with the Calendar Queue. Each bucket contains rv (the number of already reserved CNs in this bucket) and a linked list (sorted or unsorted), containing the reservations which start in this time bucket.

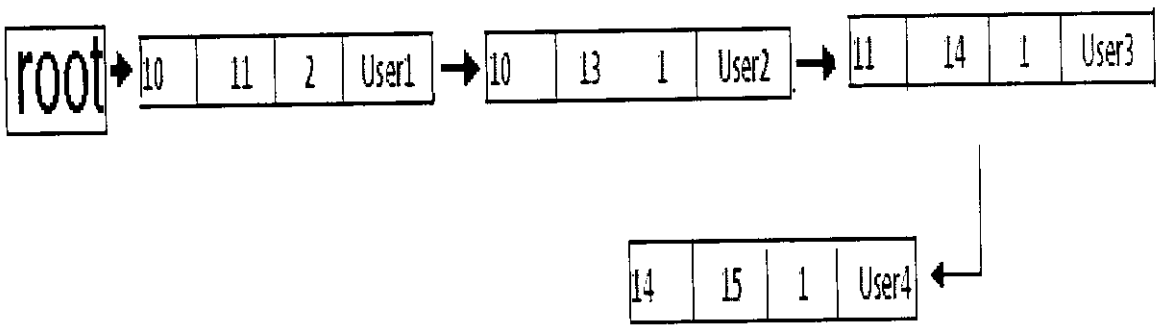
Enabling a fast O(1) access to a particular bucket, we use the following formula:

$$i = [t / sigma] \bmod M$$

Where i is the bucket index, t is the request time (in minutes), and M is the number of buckets in the data structure.

2.1.2 LINKED LIST DATA STRUCTURE

Which has a sequential searching method leading to O(totAR) time complexity, where totAR is the total number of reservations. This is because the List does not partition each reservation into a fixed time interval like a tree-based structure. That array provide better performance than a tree-based structure.



Search operation, Linked List has O(totAR m_{sub}), where tot_{AR} is the total number of reservations, and msub is the number of slots in the subinterval. The same approach also applied to the interval search operation.

Shifting the time interval to [ts+λ, te+λ] instead, where λ is the length of busy period found from the previous search operation. The interval search operation ends when it reaches the tail of the List and/or (te+λ) > (ts + MAX LIMIT), where MAX LIMIT denotes the maximum time needed for searching.

2.1.2 PROPOSED SYSTEM

2.2.1 GRID ADVANCED RESERVATION QUEUE

In this report, first describe modified versions of Linked List and Segment Tree data structures to support add, delete, and search, as well as the interval search operation capable of dealing with advance reservations in computational Grids. For this, we had to specifically develop an algorithm for finding closest interval to a requested reservation for Segment Tree.

Second, adapt Calendar Queue data structure for managing reservations as well. Calendar Queue is a priority queue for future event set (FES) problems in discrete event simulation. FES shares similar characteristics to advance reservations in Grids,

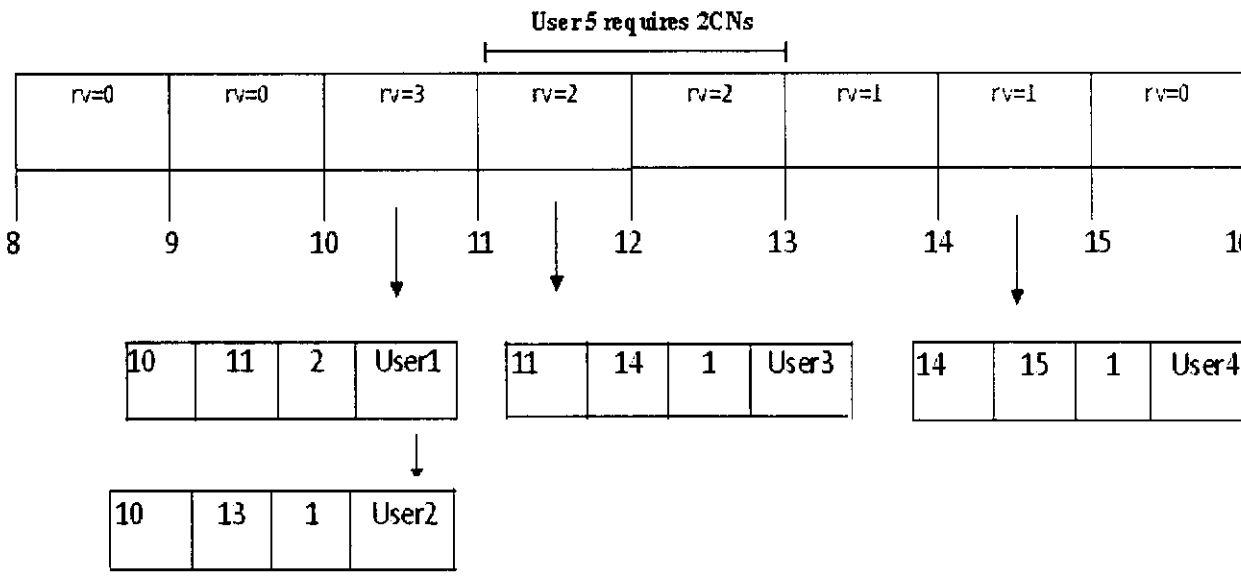
We propose a new data structure that is tailored to handle the aboveOperations efficiently. The new data structure is called Grid advanced reservation Queue (GarQ), which is a combination of Calendar Queue and Segment Tree, GarQ has a better performance time on average when dealing with reservation operations compared to other data structures.

We propose an array-based structure for managing reservations in Grid computing. The idea behind this data structure was partially inspired by Calendar Queue and Segment Tree. By combining Calendar Queue and Segment Tree into this structure. We obtain the following advantage:

1) Add new reservations directly into a particular bucket. Hence, it has a fast O(1) access to the bucket.
2) To reuse these buckets for the next time period.
3) Easy to search and compare by using iteration.

4) Easy to implement in comparison to Segment Tree and Calendar Queue
5) Flexibility in handling resource availability

Reservations are stored in .GarQ with Sorted Queue. With a sigma = 1 time interval



3. LITERATURE REVIEW

3.1 ALGORITHM FOR SCHEDULING BAG-OF-TASK APPLICATIONS

Introduction:

Grid and peer-to-peer (P2P) network technologies enable aggregation of distributed resources for solving large-scale and computationally-intensive applications. These technologies are well-suited for Bag-of-Tasks (BoT) applications, because each application consists of many parallel and independent tasks. With multiple users competing for the same resources, the key challenge is to finish a user application within a specified deadline.

They propose a time optimization algorithm that schedules a user application on auction-based resource allocation systems. These allocation systems, which are based on proportional share, allow users to bid higher in order to gain more resource shares. Therefore, this algorithm adjusts a user bid periodically on these systems in order to finish the application on time.

A Time Optimization Algorithm

In this section, two variants of a time optimization algorithm are illustrated. Before an experiment starts, each user i must specify to a broker a deadline Di on how long a user application is expected to take to complete, and an initial fund. In addition, each user i must specify a time period Pi to enable the broker to adjust the user bid on a resource r based on its share. This allows users with a tight deadline to bid more frequently. Let R be a set of n grid resources, and T be a set of m independent tasks, where each task has a variable length l. In case of a parameter-sweep application, all tasks have the same length.

Resource Discovery

A broker, who is acting on behalf of a user, contacts a GIS to get a list of available local and global resources. On each resource r, the broker gets the information of _minBidr,maxBidr, Cr, PEr_, where minBr is the minimum bidding price of r, maxBr is the maximum bidding price of r, Cr is the total processing capability of r, and PEr is the total number of Processing Element (PE) or CPU on r.

Tasks Submission

Each task t □ T is submitted to one of available resources. Moreover, tasks with smaller length l are sent to less powerfull resources if they are able to meet the deadline. If all tasks have same l, then these resources will receive a smaller quantity. Then, the remaining tasks are to other resources by using a uniform distribution. If a resource fails, then tasks are rescheduled to a different resource.

Bidding Performance

For every period P time, the broker queries the current user progress on □r □ R. Each resource r gives the broker information that contains _Sr, Br, Ltotal_, where Sr is the current user share in MI, Br is the current user bidding price, and Ltotal is the remaining total task length (including for all user tasks that are in the resource queue).

3.2 AN ARCHITECTURE FOR RESOURCE AND SCHEDULING IN GLOBAL COMPUTATIONAL GRID

Nimrod/G grid-enabled resource management and scheduling system builds on our earlier work on Nimrod and follows a modular and component-based architecture enabling extensibility, portability, ease of development, and interoperability of independently developed components. It uses the Globus toolkit services and can be easily extended to operate with any other emerging grid middleware services. It focuses on the management and scheduling of computations over dynamic resources scattered geographically across the Internet at department, enterprise, or global level with particular emphasis on developing scheduling schemes based on the concept of computational economy for a real test bed, namely, the Globus testbed (GUSTO).

This technology opportunity leads to the possibility of using networks of computers as a single, unified computing resource, popularly called cluster computing. Clusters appear in various forms: high performance clusters, high-availability clusters, dedicated clusters, non-dedicated clusters.

It is possible to cluster/couple wide varieties of geographically distributed resources such as computers including supercomputers, storage systems, data sources, and special class of devices and use them as a single unified resource, thus forming what is popularly known as "computational grids".

Computational grids are expected to popularize a model of computational economy and create a market for those who are interested in offering computational resource rental services. In such an environment, whenever an application needs additional computational resources to run, the

user can hire or rent resources on the fly and pay for what he uses. The resource price may vary from time to time and from one user to another user.

The Nimrod system has been used successfully with a static set of computational resources, but is unsuitable as implemented in the large-scale dynamic context of computational grids, where resources are scattered across several administrative domains, each with their own user policies, employing their own queuing system, varying access cost and computational power. These shortcomings are addressed by our new system called Nimrod/G that uses the Globus middleware services for dynamic resource discovery and dispatching jobs over computational grids.

It takes advantage of features supported in the latest version (v1.1) of Globus such as automatic discovery of allowed resources.

System Architecture

Its key components are:

- Client or User Station
- Parametric Engine
- Scheduler
- Dispatcher
- Job-Wrapper

The interaction between the above components and grid resources

Client or User Station

It also serves as a monitoring console and lists status of all jobs.

Another feature of the Nimrod/G client is that it is possible to run multiple instances of the same client at different locations. That means the

experiment can be started on one machine, monitored on another machine by the same or different user, and the experiment can be controlled from yet another location.

Parametric Engine

The parametric engine acts as a persistent job control agent and is the central component from where the whole experiment is managed and maintained.

The parametric engine maintains the state of the whole experiment and ensures that the state is recorded in persistent storage.

Scheduler

The scheduler is responsible for resource discovery, resource selection, and job assignment. The resource discovery algorithm interacts with a grid-information service directory (the MDS in Globus), identifies the list of authorized machines, and keeps track of resource status information.

Dispatcher

The dispatcher primarily initiates the execution of a task on the selected resource as per the scheduler's instruction. It periodically updates the status of task execution to the parametric-engine. In the current implementation, the dispatcher starts a remote component, known as job-wrapper. The job wrapper interprets a simple script containing instructions for file transfer and execution subtasks.

Job-Wrapper

The job-wrapper is responsible for staging of application tasks and data; starting execution of the task on the assigned resource and sending results back to the parametric engine via dispatcher. It basically mediates between parametric engine and the actual machine on which the task runs

3.3 A FAST O(1) PRIORITY QUEUE

A priority queue is a queue for which each element has an associated priority, and for which the dequeue operation always removes the lowest (or highest) priority item remaining in the queue. priority queue implementation for the future event set problem is described in this article. The new implementation is shown experimentally to be O(Z) in queue size for the priority increment distributions recently considered by Jones

Priority queue implementations can be classified according to the manner in which execution time varies with queue size. Let a hold operation refer to a dequeue followed immediately by an enqueue. This operation leaves the queue size unchanged. For a simple linear list implementation, the time for a hold operation is proportional to the queue size.

A priority queue implementation for the event set problem which has relatively low overhead, and which on the basis of intuition and experimental evidence has O(1) average performance. The proposed data structure is called a calendar queue. It is modeled after a desk calendar, which is the means by which a human being solves the problem of ordering a future event set.

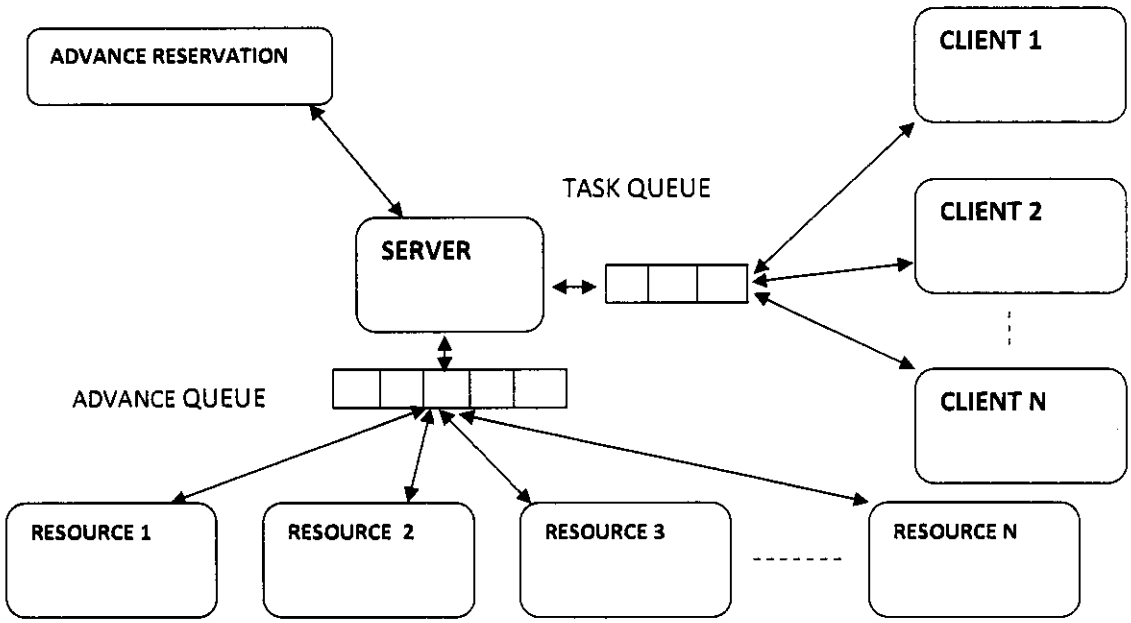
One schedules an event on a desk calendar by simply writing it on the appropriate page, one page for each day. There may be several events scheduled on a particular day, or there may be none. The time at which an event is scheduled is its priority. The enqueue operation corresponds to scheduling an event. The earliest event on the calendar is dequeued by scanning the page for today's date and removing the earliest event written on that page.

4. IMPLEMENTATION

4.1 CONSTRUCTION OF GRID

Grid infrastructures aim at giving access to resources in a secure and fair manner. In addition to creating processes on users' behalf thanks to local resource managers, infrastructures enforce access control, authenticate users and manage data movement and replication between organizations that participate to grids. Grid computing is becoming a critical component of science, business, and industry. Making grids easy to use could lead to advances in fields ranging from industrial design to systems biology to financial management. With computing cycles plentiful and inexpensive, practical grid computing would open the door to new models for compute utilities, a service similar to an electric utility in which a user buys computing time on-demand from a provider. When the Grid system initializes itself to solve a problem, it automatically retrieves from the grid a list of available computer nodes. It also obtains the grid's performance characteristics. At run-time, Optimal Grid measures ongoing performance, including communication time, computation time, and the complexity of the problem pieces. Grid uses this information to configure the grid by calculating the optimal number of computer nodes, partitioning the problem, and distributing its pieces in a way that obtains the best possible performance on whatever grid is used. Grid is a new programming model designed for the grid environment. It is optimal in the sense that the system attempts to optimize and balance the pieces of the workload to make the best use of any existing grid infrastructure. Initial results look promising.

The over all system architecture show the Grid Advance Resource Reservation Queue. The following pictorial representation show process of GARQ System, here grid network consist of set resources and client.



Proposed system Architecture

4.2 IMPLEMENTATION OF GRID ADVANCED RESERVATION QUEUE

Advance Reservation (AR) is the process of requesting resources for use at specific times in the future. Common resources that can be reserved are compute nodes and network bandwidth. AR in a scheduling system solves the above problem by allowing users to gain simultaneous and concurrent access to adequate resources for the execution of such applications. In order to reserve the available resources in such AR systems, a user must first submit a request by specifying a series of parameters such as number of compute nodes needed, and start time and duration of jobs. Then, the AR system checks for the

feasibility of this request. If there are no available nodes for this requested time period, the request is rejected. Consequently, the user may resubmit a new request with a different start time and/or duration until available nodes can be found. Given this scenario, the choice of efficient data structure can significantly minimize the time complexity needed to search available compute nodes, add new requests, and delete existing reservations.

4.3 SEARCHING FOR AVAILABLE COMPUTING NODES(CNS)

Each queue has a finite buffer with size S to store objects waiting to be processed by one of P independent CNs. All CNs are connected by a high-speed network. The CNs in the resource can be homogeneous or heterogeneous. In this paper, we assume that a resource has homogeneous CNs, each having the same processing power, memory and hard disk. Therefore, the primary role of the data structure is to store and to update the information about CNs' availability as time progresses. Then, a resource scheduler is responsible for managing incoming jobs and assigning them to available CNs.

4.4 PROVISIONING FOR ADDING AND DELETING OF RESERVATION

The reservation model uses a two-phase commit and gives the user a chance to negotiate with the RS if the request gets rejected. Hence, in order to support this reservation scenario, a data structure needs to perform the following basic operations: Search: checking for availability of CNs in a given time interval. Add: inserting a new reservation request into the data structure. This operation is performed only when the previous search phase succeeded. Delete: removing the existing reservation from the data structure. This operation is conducted only when the add phase succeeds and the reservation's finish time has passed.

4.5 SEARCHING FOR A FREE SLOT

- $N_{availableCN}$ is number of available CNs in the whole interval of the node N;
- $leftS, rightS$ are temporary variables, that store the suggested starting time from the left and right subtree respectively; and

Δ is the length of the reservation interval, so simply $\Delta = r - l$.

4.6 PERFORMANCE COMPARISON

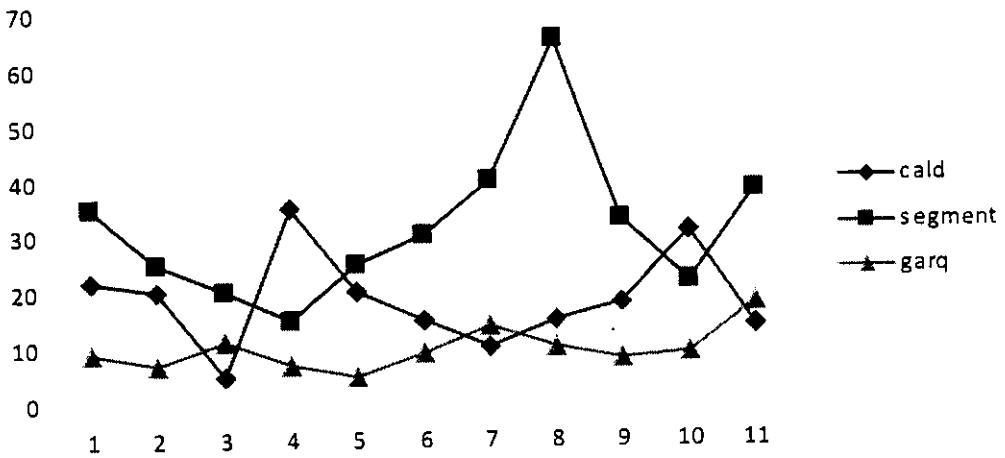
Thus the performance of the proposed system is compared with the performance of the existing system.

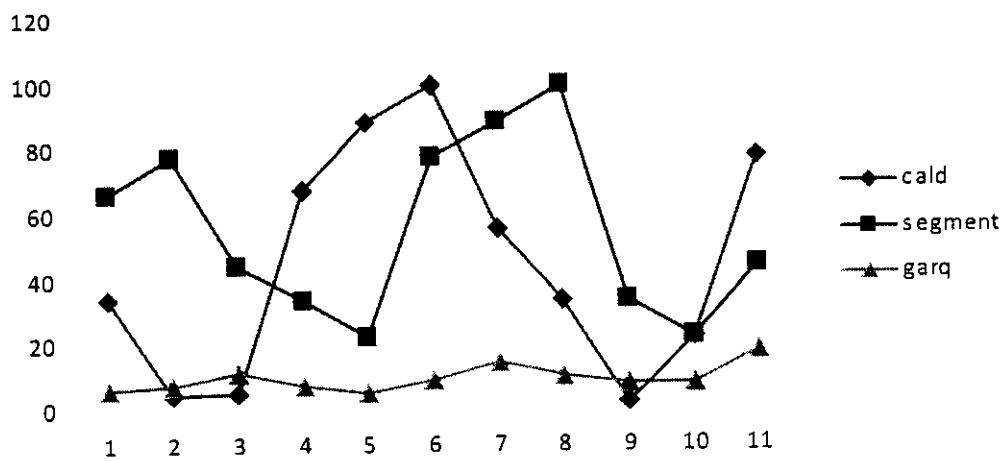
In order to evaluate the performance of our proposed data structure, i.e. GarQ with Unsorted Queue (GarQ-U) and GarQ with Sorted Queue (GarQ-S), we compare them to Linked List (List), Segment Tree (Tree) with slot length = 5 minutes, and static Calendar Queue (SCQ) with δ = 1 hour. For SCQ to be optimal, we choose the value of δ based on the jobs' average duration time as stated in Table 1. For GarQ-U and GarQ-S, we set each slot to be a 5-minute interval. All, except List, have a fixed interval.

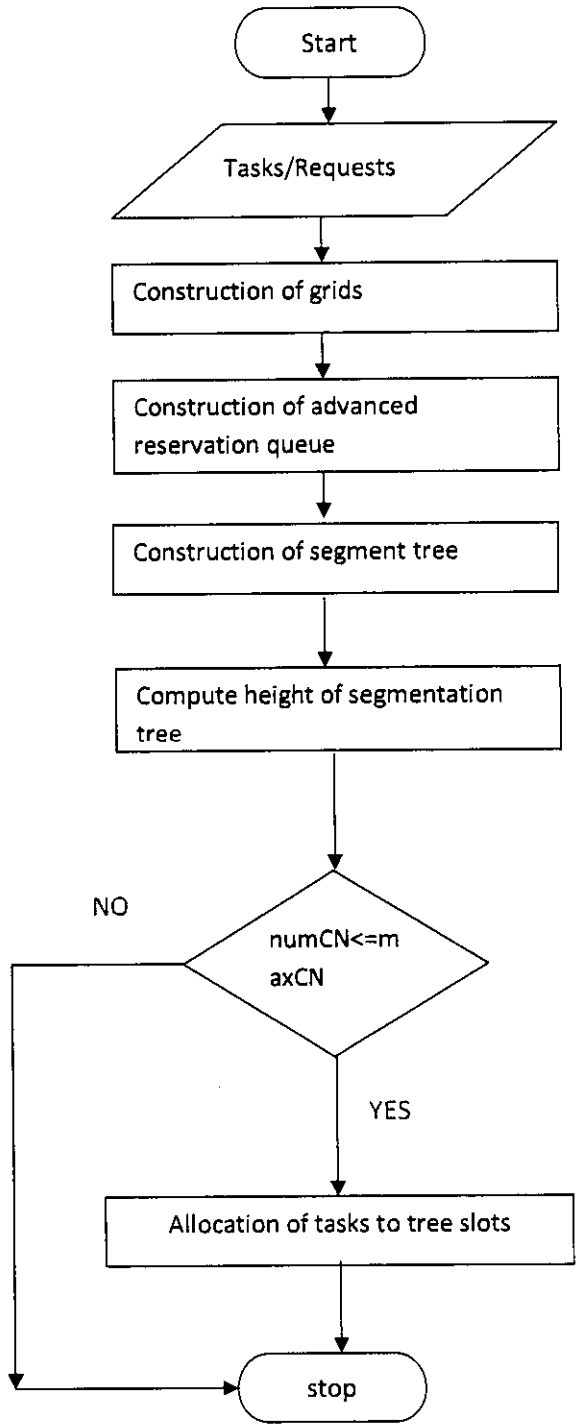
Table 1:

Name	Time Complexity		
	Add	Delete	Search
Segment Tree	O(log n)	O(log n)	O(log n)
Calendar Queue	O(k)	O(1)	O(k m_{sub})
Linked Lst	O(tot AR)	O(tot AR)	O(tot AR m_{sub})
GarQ with Unsorted Queue	O(m_{sub})	O(k+m_{sub})	O(m_{sub})

Total number of nodes accessed during *add* and *delete* operation using original traces






Start
Tasks/Requests
Construction of grids
Construction of advanced reservation queue
Construction of segment tree
Compute height of segmentation tree
NO
numCN<=maxCN
YES
Allocation of tasks to tree slots
stop

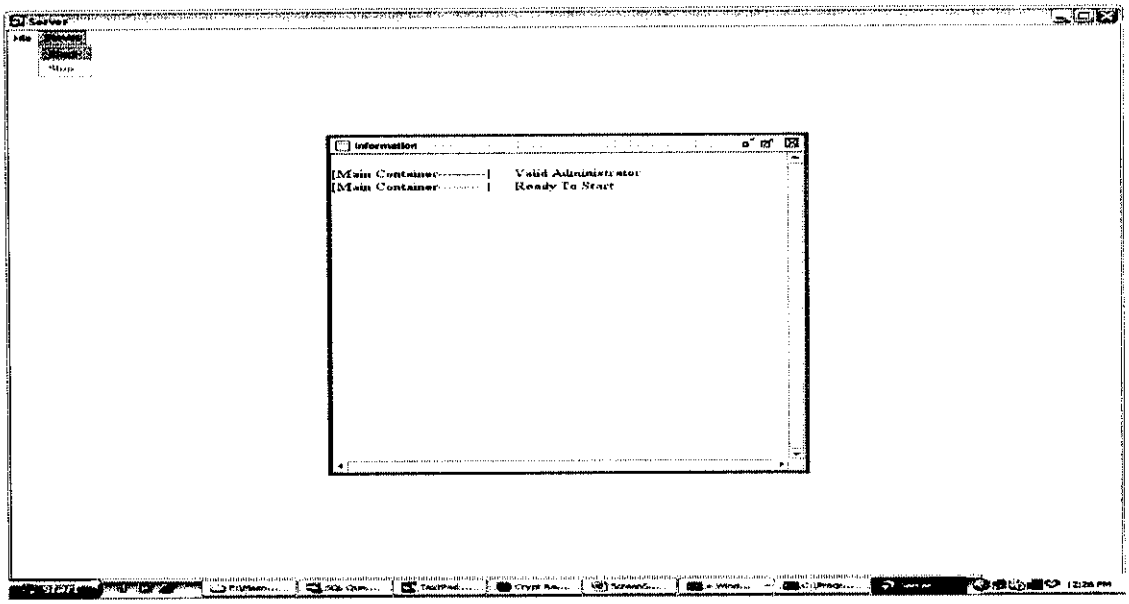
5. CONCLUSION AND FURTHER WORK

Advance Reservation (AR) in Grid computing is an important research area as it allows users to gain concurrent access to resources by allowing their applications to be executed in parallel. It also provides guarantees on the availability of resources at the specified times in the future. An efficient data structure is significant in minimizing the time complexity needed to perform AR operations. In this work presented a new data structure, they named it GarQ, to efficiently search for available compute nodes, to add new requests, and to delete existing reservations. It also introduced a new operation called interval search for finding a free interval closest to the requested reservation, if it was previously rejected. This operation is important because it helps users in negotiating a suitable reservation time.

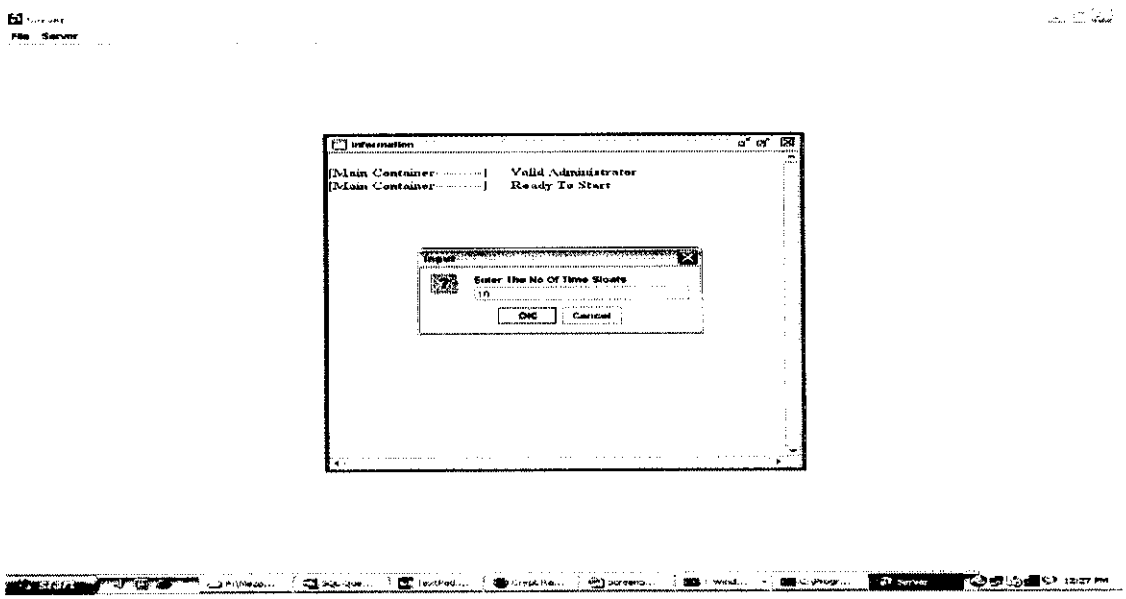
FURTHER WORK

An extension to this work is to consider imposing a minimum duration limit by a resource and/or grouping small jobs as one big batch before requesting a reservation. With this approach, GarQ will be able to perform more efficiently since this scenario will be similar to reserve large jobs. Moreover, we are thinking of comparing the performance and effectiveness of GarQ and other data structures when dealing with the affects of scheduling issues, such as deadline, backfilling and job preemption. Finally, different types of traces and applications need to be considered in the performance evaluation.

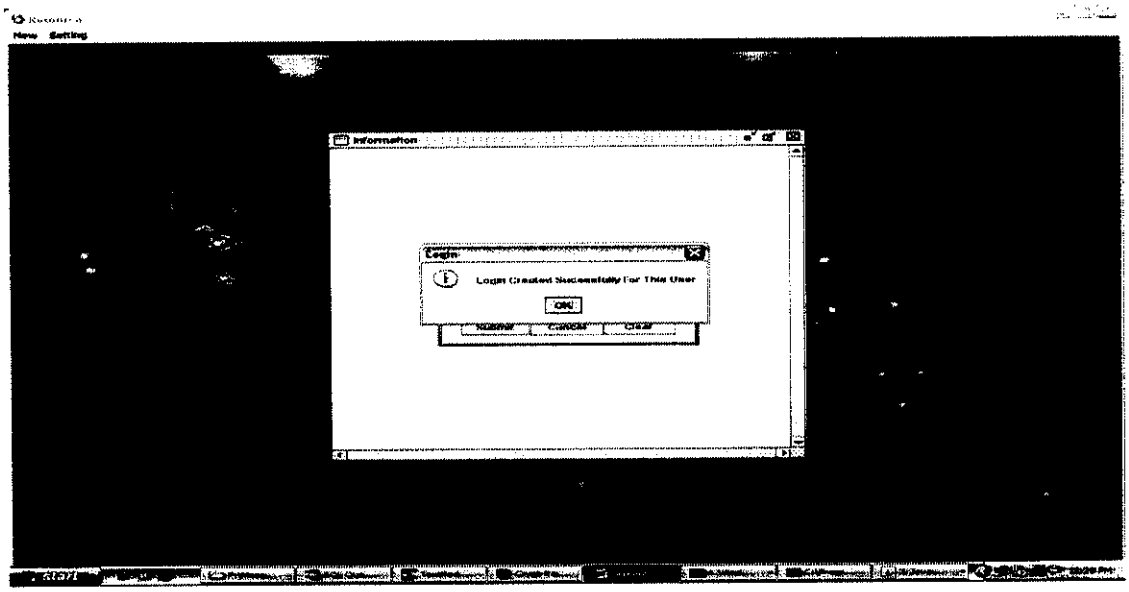
6. APPENDIX

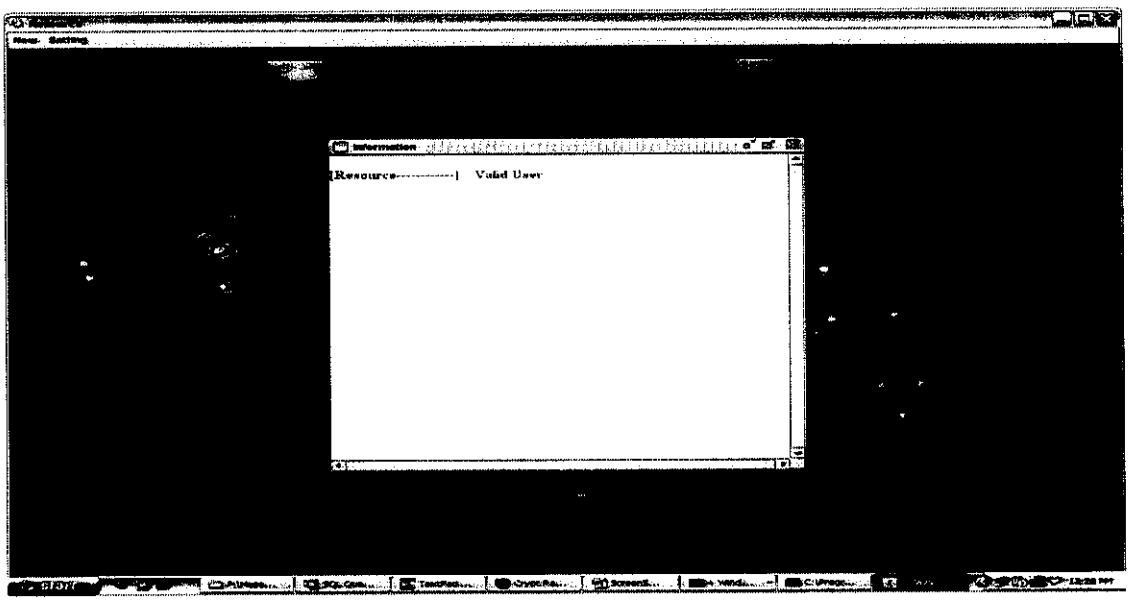


SERVER ALLOCATE THE TIME SLOTS

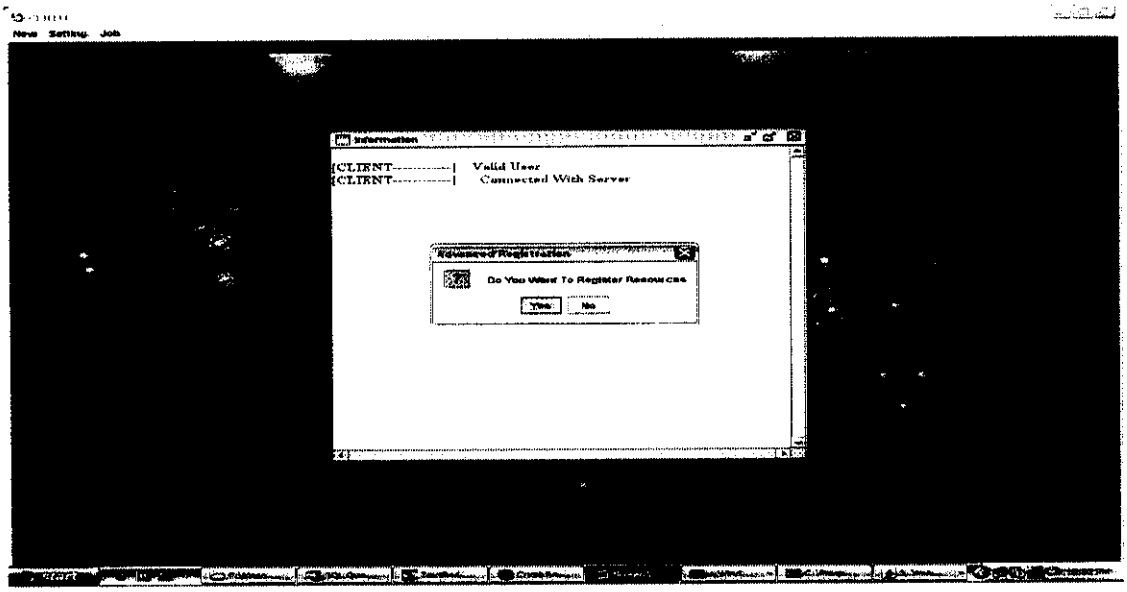


RESOURCE ALLOCATE TO USER

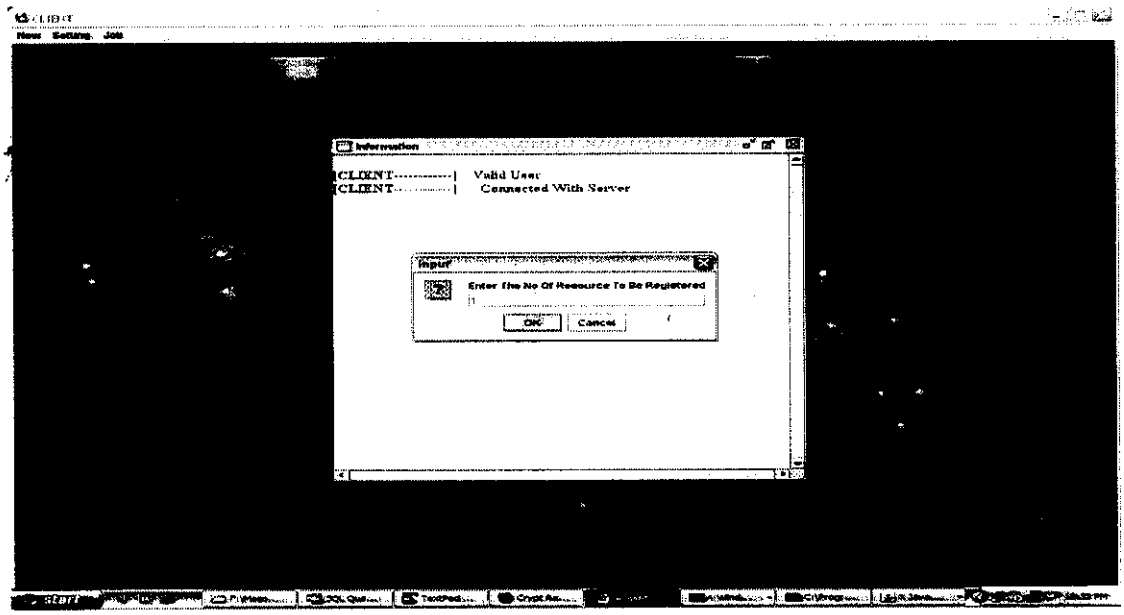




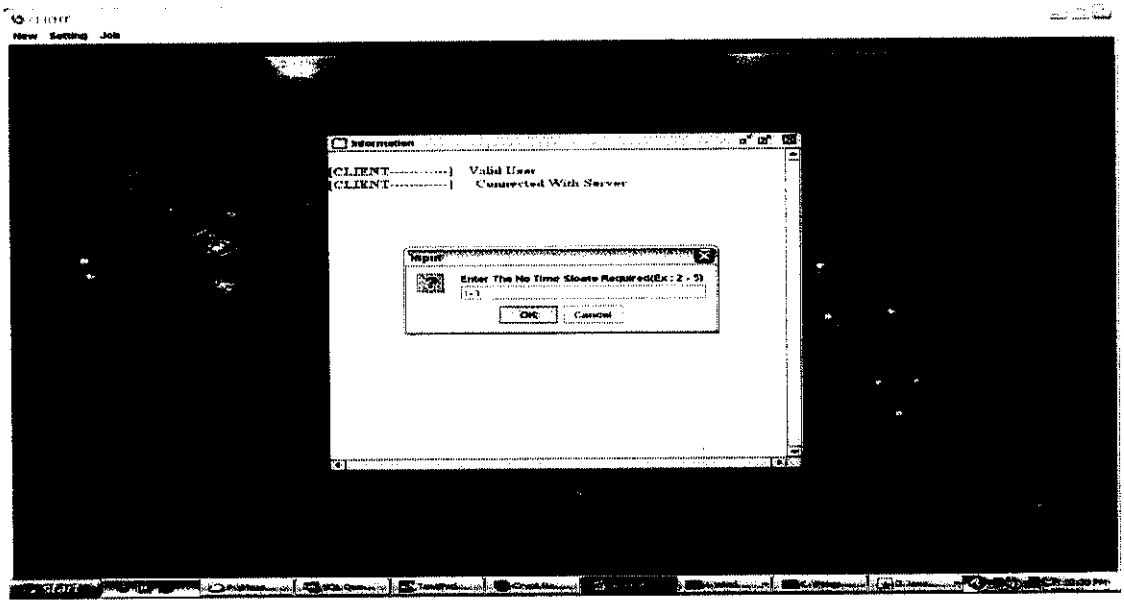
ADVANCE RESOURCE TO USER



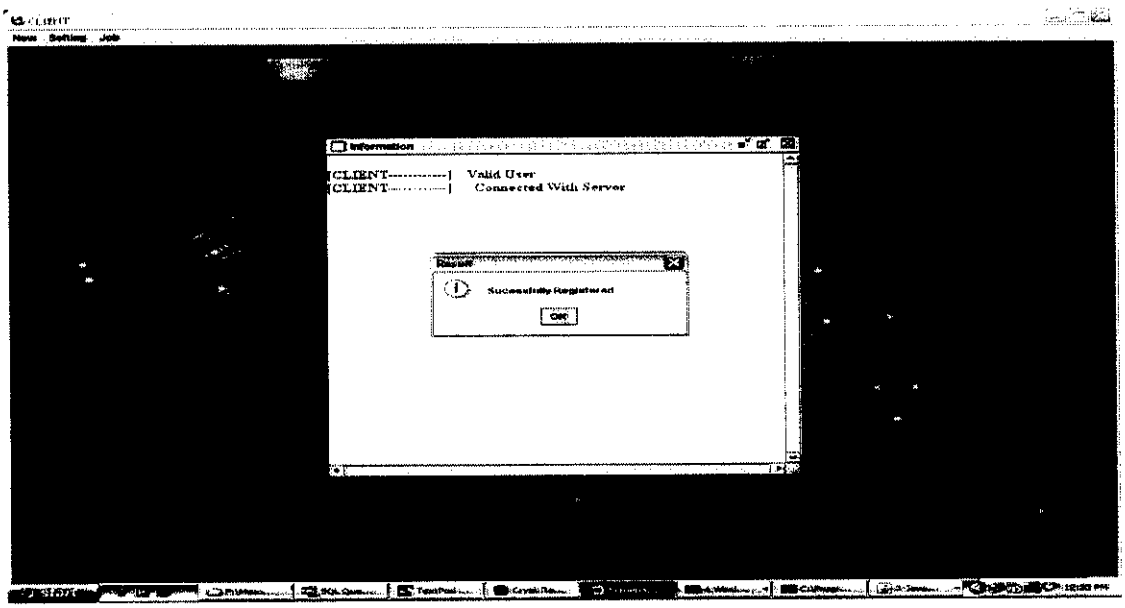
NUMBER OF RESOURCE TO USER



REQUIRED TIME SLOTS TO USER



RESOURCE ALLOCATED TO USER



CODING

CLIENT

```
import java.io.*;
import java.net.*;
import java.rmi.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.sql.ResultSet;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.StringTokenizer;
import java.util.ArrayList;

class Client extends JFrame implements ActionListener
{
JMenuBar mbar;
JMenu m1,m2,m3;
JMenuItem mi,mi1,mi2,mi3,mi4,mi5;
String hostname="127.0.0.1";
String path="";
database_conn db=null;
MyDesktopPane desktop;
Object obj[]=new Object[5];
Serverinterface bin=null;
```

```
public static String d="";
public static boolean loop=true;
public ConnectClient CC=null;
public JFrame frFrame=null;
ArrayList reg = null;

public Client()
{
super("Client");
frFrame=this;
mbar=new JMenuBar();
m1=new JMenu("New");
m2=new JMenu("Setting");
m3=new JMenu("Job");
mi4=new JMenuItem("New User");
mi=new JMenuItem("Login");
mi1=new JMenuItem("Connect");
mi2=new JMenuItem("Exit");
mi3=new JMenuItem("Server IP");
mi5=new JMenuItem("Job");
reg = new ArrayList();
m1.add(mi4);
m1.add(mi);
m1.add(mi1);
m1.add(mi2);
m2.add(mi3);
m3.add(mi5);
mbar.add(m1);
mbar.add(m2);
mbar.add(m3);
setJMenuBar(mbar);
mi1.setEnabled(false);
mi3.setEnabled(false);
mi5.setEnabled(false);
mi.addActionListener(this);
```

```
mi1.addActionListener(this);
mi2.addActionListener(this);
mi3.addActionListener(this);
mi4.addActionListener(this);
mi5.addActionListener(this);
obj[0]=mi;
obj[1]=mi1;
obj[2]=mi3;
frFrame.setTitle("CLIENT");
desktop=new MyDesktopPane();
setContentPane(desktop);
Image ic=Toolkit.getDefaultToolkit().getImage("Images/i3.gif");
setIconImage(ic);
//setIconImage(getToolkit().getImage("Images/image3.jpeg"));
Dimension ss=Toolkit.getDefaultToolkit().getScreenSize();
setSize(ss.width,ss.height);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
db=new database_conn();
textArea txtarea=new textArea();
display(txtarea);
try
{
path=new java.io.File(".").getCanonicalPath();
ResultSet rs=db.stat.executeQuery("Select * from clientlogin");
boolean test=false;
while(rs.next())
test=true;
if(!test)
mi.setEnabled(false);
}
catch(Exception e)
{
System.out.println("Error Occured While Reading The DataBase
:\n"+e);
```

```
}
}
public void actionPerformed(ActionEvent ae)
{
Object src=ae.getSource();

if(src.equals(mi))
{
LoginClient lc=new LoginClient(obj);
display(lc);
}
if(src.equals(mi4))
{
NewUser nu=new NewUser(obj);
display(nu);

}
if(src.equals(mi1))
{

CC=new ConnectClient(hostname);
mi5.setEnabled(true);
}
if(src.equals(mi2))
{
System.exit(0);

}
if(src.equals(mi3))
{
hostname=JOptionPane.showInputDialog("Enter The Server IP
:","127.0.0.1");
mi1.setEnabled(true);
}
if(src.equals(mi5))
```

```
{

}
}

void display(JInternalFrame obj)
{
new CenterFrame(obj);
obj.setVisible(true);
desktop.add(obj);
try
{
obj.setSelected(true);
}
catch(java.beans.PropertyVetoException e2)
{
e2.printStackTrace();
}

}
public static void main(String[ ] args)
{
new Client();

}

}
```

RESOURCE ALLOCATION

```
import java.io.*;
import java.net.*;
import java.rmi.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.sql.ResultSet;
import java.util.ArrayList;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.StringTokenizer;

class ConnectResource implements Serializable
{
Serverinterface bin=null;
public static String d="";
public static boolean loop=true;
SerializedObject so=null;
String hostname="127.0.0.1";
String host="";
String path="";
public static String type="Resource";

ConnectResource(String hostname)
{
try
```

```
{
this.hostname=hostname;
path=new java.io.File(".").getCanonicalPath();
System.out.println("rmi://"+hostname+"/StartServer");
bin=(Serverinterface)Naming.lookup("rmi://"+hostname+"/StartServe
r");
System.out.println("[Resource----------]"+"  Connected with server
:\n");
textArea.setTxt("[Resource----------]"+"    "+"  Connected With
Server");
InetAddress addr = InetAddress.getLocalHost();
// Get IP Address
byte[] ipaAddr = addr.getAddress();
host=addr.getHostName();
so = new SerializedObject();
so.setResource(this);
bin.connect(so);
/*
while(true)
{

InputStreamReader ir=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(ir);
String str1=br.readLine();
String st=b.send(str1);
System.out.println("from server:"+st);
}
//*/
}
catch(Exception e)
{
e.printStackTrace();
}

}
```

```
public void register(ArrayList param)
{
try
{
for(int i=0;i<param.size();i++)
{
String str=param.get(i).toString();
System.out.println("From RC : "+str);
}
SerializedObject so = new SerializedObject();
so.setResource(this);
so.setArrayList(param);
String provider=bin.reqResource(so);
textArea.setTxt("[CLIENT----------]"+"    "+" Request Form Sent
Successfully");
if(provider.equals("Provider Not Avalible"))
{
JOptionPane.showMessageDialog(null,provider,"Result",JOptionPane
.INFORMATION_MESSAGE);
textArea.setTxt("[CLIENT----------]"+"    "+" Result Receive
Successfully : [ "+provider+" ]");
}
else
{
JOptionPane.showMessageDialog(null,"Available Client Is :
"+provider,"Result",JOptionPane.INFORMATION_MESSAGE);
textArea.setTxt("[CLIENT----------]"+"    "+" Result Receive
Successfully : [ Selected Provider Is : "+provider+" ]");
}

}
catch(Exception e)
{
e.printStackTrace();}}}
```

SERVER

```
import java.io.*;
import java.rmi.server.*;
import java.rmi.*;
import java.util.*;
import javax.swing.*;
import java.sql.ResultSet;

class StartServer extends UnicastRemoteObject implements
Serverinterface
{
static String str;
public static Vector client=new Vector();
public static int clientCount;
public static Vector resource=new Vector();
public static int resourceCount;
public static int timeSloat[]=null;
public static int Rno=10;
public static String resClient[]=null;
String host = "";
database_conn db=null;
int index[];
int resindex[];
public StartServer()throws RemoteException
{
clientCount=0;
resourceCount=0;
db=new database_conn();
String input=JOptionPane.showInputDialog("Enter The No Of Time
Sloats","10");
Rno = Integer.parseInt(input);
timeSloat = new int[Rno];
String query = "create table mainTable(resource varchar(50),";
for(int i=1;i<=Rno;i++)
```

```
{
timeSloat[i-1]=i;
if(i<Rno)
query +="\""+i+"\" int,";
else
query += "\""+i+"\" int)";
}
try
{
db.stat.execute("if object_id('server') is not null drop table server");
db.stat.execute("create table server(resourceip varchar(50),clinetip
varchar(50),timeSloat varchar(20))");
db.stat.execute("if object_id('resource') is not null drop table
resource");
db.stat.execute("create table resource(resourceip varchar(50))");
db.stat.execute("if object_id('client') is not null drop table client");
db.stat.execute("create table client(clientip varchar(50))");
db.stat.execute("if object_id('mainTable') is not null drop table
mainTable");
db.stat.execute(query);
}
catch(Exception e)
{
e.printStackTrace();
}
}

public void connect(SerializedObject so1)throws RemoteException
{
ConnectClient C1=so1.getClient();
ConnectResource R1=so1.getResource();
if(C1!=null)
{
client.add(C1);
clientCount++;
```

```
this.host = C1.host;
System.out.println("Connected Client No : "+clientCount+"");
textArea.setTxt("[Main Container----------]"+"  Connected Client :
"+C1.host+"");
try
{
String table = C1.host+"client"+clientCount;
db.stat.execute("if object_id('"+table+"') is not null drop table
"+table);
db.stat.execute("create table "+table+"(resourceip
varchar(50),timeSloat varchar(20))");
db.stat.execute("insert into client values('"+table+"')");
System.out.println("Table Created For Table Name : "+table);
int rc=0;
String ts=null;
do
{
String input=JOptionPane.showInputDialog("Enter The No Of
Resource To Be Registered","Between 1 To "+resourceCount);
rc=Integer.parseInt(input);
if(rc <= resourceCount)
{
ts=JOptionPane.showInputDialog("Enter The No Time Sloats
Required(Ex : 2 - 5) ","Between 1 to "+Rno);
break;
}

}
while(true);
System.out.println("No Of Resources Recqired : "+rc);
System.out.println("Required Time Sloat : "+ts);
String tst[]=ts.split("-");
resClient = new String[resourceCount];
int ra=0;
boolean suce = true;
```

```
boolean test[]= new boolean[rc];
for(int i=0;i<rc;i++)
{
test[i]=true;
int s = Integer.parseInt(tst[0].trim());
int e = Integer.parseInt(tst[1].trim());
for(int j = s;j <= e;j++ )
{
ResultSet rs = db.stat.executeQuery("select * from mainTable");
if(rs.next())
{
int k=0;
do
{
resClient[k] = rs.getString(1);
k++;
//System.out.println("\""+j+"\"");
int a = rs.getInt(""+j+"");
if(a == 0)
{
db.stat1.executeUpdate("update mainTable set \""+j+"\" = 1 where
resource = '" +resClient[i]+"'");
}
else
{
for(int l = i-1;l >= s;l++ )
db.stat.executeUpdate("update mainTable set \""+l+"\" = 0 where
resource = '" +resClient[i]+"'");
test[i] = false;
break;
}
}
while(rs.next());
}
if(test[i])
```

```
{
suce = false;
break;
}
}


}
//if(suce)
JOptionPane.showMessageDialog(null,"Sucessfully
Registered","Result",JOptionPane.INFORMATION_MESSAGE);
//else
//JOptionPane.showMessageDialog(null,"Registered Failed Resurce
Not Avaliable
Waiting...","Result",JOptionPane.INFORMATION_MESSAGE);


}
catch(Exception e)
{
System.out.println("Error While Creating Table For The Client");
e.printStackTrace();
}
}
else if(R1!=null)
{
resource.add(R1);
resourceCount++;
System.out.println("Connected Resource No : "+resourceCount+"");
textArea.setTxt("[Main Container----------]"+"  Connected Resource :
"+R1.host+"");
try
{
String table = R1.host+"res"+resourceCount;
```

```
db.stat.execute("if object_id('"+table+"') is not null drop table
"+table);
db.stat.execute("create table "+table+"(clientip varchar(50),timeSloat
varchar(20))");
db.stat.execute("insert into resource values('"+table+"')");
db.stat.execute("insert into mainTable(resource) values('"+table+"')");
System.out.println("Table Created For Table Name : "+table);
}
catch(Exception e)
{
System.out.println("Error While Creating Table For The Client");
e.printStackTrace();
}
}
}
public void registerClient(SerializedObject so1)throws
RemoteException
{
ConnectClient C1=so1.getClient();
Random r=new Random();
int a1[]={1,0};
String table = C1.host+"client"+clientCount;
String query = "insert into "+table+" values(";
try
{
ArrayList a=so1.getArrayList();
for(int i=0;i<a.size();i++)
{
String str=a.get(i).toString();
query +="'"+str+"',";
}
query+=a1[r.nextInt(a1.length)]+")";
System.out.println(query);
db.stat.execute(query);
```

```
textArea.setTxt("[Main Container----------]"+"  Received Data From
Agent : "+C1.host+"");
}
catch(Exception e)
{
System.out.println("Error While Creating Table For The Client");
e.printStackTrace();
}
}
public String reqResource(SerializedObject so1)throws
RemoteException
{
ConnectResource CR1=so1.getResource();
String table = CR1.host+"res"+resourceCount;
String restbl = CR1.host+"res"+resourceCount+"result";
String query = "insert into "+table+" values(";
try
{
db.stat.execute("if object_id('"+restbl+"') is null create table
"+restbl+"(sno int IDENTITY (1,1), client varchar(50))");
ArrayList a=so1.getArrayList();
for(int i=0;i<a.size();i++)
{
String str=a.get(i).toString();
if(i == a.size()-1)
query +="'"+str+"'";
else
query +="'"+str+"',";
}
query+=")";
//System.out.println(query);
db.stat.execute(query);
textArea.setTxt("[Main Container----------]"+"  Received Data From
Buyer : "+CR1.host+"");
String res = "";//result(table,restbl);
```

```
System.out.println(res);
textArea.setTxt("[Main Container----------]"+"  Result Sent To Buyer :
"+CR1.host+"");
return res;

}
catch(Exception e)
{
System.out.println("Error While Creating Table For The Client");
e.printStackTrace();
}
return "Provider Not Avalible";

}

public String result(String table,String restbl)
{

String ret = "Provider Not Avalible";
/*new Baysian(clientCount,host);
try
{
ResultSet rs,rs1;
rs = db.stat.executeQuery("select volume,price,duedate from "+table);
if(rs.last());
for(int i=0;i<clientCount;i++)
{
String tbl = "probclient"+(i+1);
rs1 = db.stat1.executeQuery("select volume,price,duedate from
"+tbl+" where trust = 1 order by prob desc");
if(rs1.next())
{
do
{
boolean test = false;
```

```
for(int j=1;j<=3;j++)
{
double a = rs.getDouble(j);
double b = rs1.getDouble(j);
System.out.print(a+" , "+ b);
if(a == b)
{
test = true;
}
else
{
test = false;
break;
}
}
if(test)
{
ret=host+"client"+(i+1);
break;
}
System.out.println();
}
while(rs1.next());
}

}
db.stat.execute("insert into "+restbl+" values('"+ret+"')");
return ret;

}
catch(Exception e)
{
e.printStackTrace();
}
                                    */return ret;}}
```

7. REFERENCES

1) D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. Communications of the ACM, 45(11):56.61, 2002.

2) A. Brodnik and A. Nilsson. A static data structure for discrete advance bandwidth reservations on the internet. In Proc. of Swedish National Computer Networking Workshop (SNCNW), Stockholm, Sweden, September 2003.

3) R. Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. Communications of the ACM, 31(10):1220.1227, 1988.

4) R. Buyya, D. Abramson, and J. Giddy. Nimrod-G: An architecture for a resource management and scheduling system in a global computational grid. In Proc. of the 4th Intl. Conference & Exhibition on High Performance Computing in Asia-Paci_c Region (HPC Asia), Beijing, China, May 2000.

5) W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In Proc. of the International Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico, May 1.5 2000.

6) Q. Xiong, C. Wu, J. Xing, L. Wu, and H. Zhang. A linked-list data structure for advance reservation admission control. In Proc. of the 3rd International Conference on Networking and Mobile Computing (ICCNMC), Zhangjiajie, China, August 2-4 2005.

7) K. B. Erickson, R. E. Ladner, and A. Lamarca. Optimizing static calendar queues. ACM Trans. on Modeling and Computer Simulation, 10(3):179.214, 2000.

8) L. Yuan, C.-K. Tham, and A. L. Ananda. A probing approach for effective distributed resource reservation. In Proc. of the 2nd International Workshop on Quality of Service in Multiservice IP Networks, pages 672–688, Milan, Italy, February 2003. Springer-Verlag.

9) L.-O. Burchard. Analysis of data structures for admission control of advance reservation requests. IEEE Transactions on Knowledge and Data Engineering, 17(3), 2005.