DESIGN AND DEVELOPMENT OF A TICKET-BASED SCHEDULING AND LOAD SHARING ALGORITHM FOR OPTIMAL RESOURCE USAGE IN MOBILE COMPUTING ENVIRONMENTS

Natarajan Meghanathan[1] and Sanjeev Baskiyar[2]

[1]Jackson State University, Jackson, MS, USA [2]Auburn University, Auburn, AL, USA

[1]Corresponding Author E-mail: natarajan.meghanathan@jsums.edu

ABSTRACT

Clients in mobile cellular networks lack computing resources to execute complex and data-intensive applications. So the mobile clients purchase tickets from ticket engines installed at stationary servers that are rich in resources. Clients upload computation, resource intensive jobs or applications along with the ticket to the stationary servers. The tickets store the client validity information, deadline before which the stationary servers should execute the job and forward the results to the destination cell, the priority of execution of the job and the run-time history information about the jobs (like the amount resources the job required when it was run before). For simplicity, the stationary servers are assumed to be the base stations. The base stations then schedule the jobs based on the information in the ticket. If the base station at which a job is submitted determines that it will not be able to execute and forward the job to the destination cell before the deadline, the base station forwards it to a supervisory host which can be compared to a base station controller in GSM networks. The supervisory host then schedules the job to any peer base station which can handle the job and forward to the destination before the deadline. The design was simulated using Java. The servers were simulated as shared memory multiprocessor systems. It was determined that on the average there is a 15-60% decrease in turn around time for the jobs executed based on the ticket-based scheduling and load sharing (TB-SLS) model when compared with that of the FIFO or RR models. Also, the percentage deadline guarantee with the ticket model was 30-65% while that of the FIFO and RR models is lower. The simulation was performed on both homogeneous and heterogeneous systems of servers and clients.

KEYWORDS

Ticket, Resource Scheduling, Load Sharing, Mobile Computing, Algorithm Design

1. INTRODUCTION

Two important trends of network communication are currently visible [18]. The network service model in the realm of wired networks is evolving from a best-effort-only model into a scalable multiple service model and hence offering the possibility of significantly enhanced performance to network applications. Simultaneously wireless and mobile communication networks are becoming more robust and ubiquitous. Even though many researches have addressed these two trends individually, more research is required to address the convergence of these two trends. Wireless mobile networks are characterized with frequent network disconnections, widely varying bandwidth between wired and wireless links, frequent server hand-off and limited

computing power of the mobile devices. The problem of resource scheduling and load sharing in mobile computing environments is more complicated than that in distributed systems [10]. Mobile computers change their position from time to time, and their platforms are not only heterogeneous but also dynamic.

Traditional methods of resource scheduling and load sharing cannot be employed in mobile computing environments without the support of new software or changes. The differences between the hardware characteristics of the mobile computers and those of stationary ones suggest that mobile computers should take advantage of the resources of the stationary computers [10]. To achieve better performance, mobile computers should delegate tasks to more powerful or lightly loaded stationary hosts. Also, one can obtain better responses when computationally intensive applications are executed on a more powerful foreign stationary host than on the mobile computer itself. Communication links between mobile computers in a wireless network are slow and unreliable [11]. Performance can be improved if communication between mobile computers can be reduced. For example, consider a case where a long job on mobile computer M_1 has to often communicate with another mobile computer M_2 via stationary hosts S_1 and S_2. To reduce communication between M_1 and S_1, the job may be submitted to run on S_1 instead of M_1.
The use of tickets to access various services has become as a well-established concept in everyday life. A ticket is a piece of information that represents a user's rights to access a certain service provided by a certain service provider. This general concept of tickets can be extended for service access in the context of communication systems. But, however the physical realization of tickets in this case is different: a string of bits [5]. The following are the main advantages of using tickets for accessing services [5]: (i) ***Flexibility:*** Clients can choose services that in a way satisfy their needs and match their personal requirements. Hence, by buying appropriate set of tickets, one can easily construct personalized service profiles for the clients. Also, clients need not engage in long-term contractual relationships with service providers. (ii) ***Scalability:*** A ticket can contain all the necessary information that would enable a service provider to decide whether it should or be able to provide the service or not. In other words, service providers need not run long distance protocols with some trusted agent of the accessing user (for authentication). Thus a ticket-based service approach is more suitable for mobile communication systems where users roam and contact service providers in foreign domains. (iii) ***Privacy:*** A user only needs to demonstrate that he is a legitimate holder of the ticket and need not reveal his real identity. A trusted agent can issue tickets and handle payments on behalf of users. Thus, a ticket-based mechanism along with a trusted agent can protect the privacy of the users efficiently.

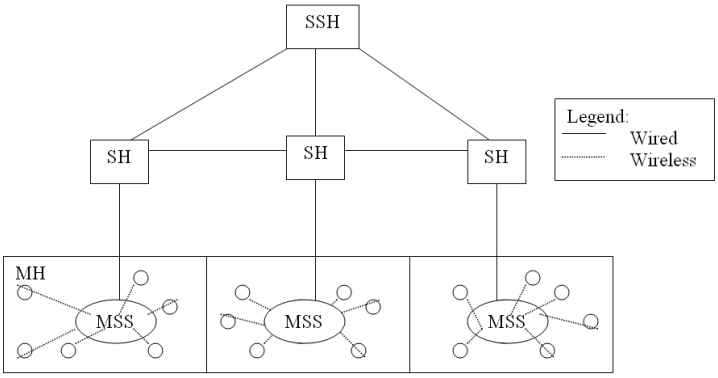

Figure 1: Four-layered Mobile Cellular Network Architecture

This paper proposes a broader use of tickets, than in earlier works, for resource scheduling, load sharing and billing in mobile computing environments. Tickets combine the features of credit card and workload information submitted to the service provider. We consider a four-layered micro-cellular network architecture, as shown in Figure 1, consisting of Mobile Hosts (MHs), Mobile Support Stations (MSSs), Supervisory Hosts (SHs) and a Super Supervisory Host (SSH). The single SSH manages a group of SHs, which in turn manage a group of MSSs, also called base stations, with each cell having an MSS. Clients buy tickets from the MSS for service and resource access. Each MSS is assumed to be a shared memory multi processor system. MHs (also called clients) upload tickets with their database and computation-intensive applications and submit to an MSS. More than one MH can run the same application (for e.g. internet search). A job refers to an instance of an application. The jobs at an SSH, SH and MSS may be respectively from a peer SH, peer MSS and MH. All job executions are assumed to occur only at the MSSs. The SSH and SHs are involved only in resource scheduling and load sharing.

2. LITERATURE REVIEW

Mobile computing is an emerging computing environment that incorporates both wireless and high-speed networking technologies [4]. Users equipped with mobile devices like personal digital assistants (PDAs) and palm-top computers will have access to a wide variety of services available over several national and international communication networks. To receive or access these services, mobile users need to be connected to wired (fixed) networks through wireless (mobile) networks. In this section, we review several works focused towards the convergence of these two networks and that would lead to a generalized service architecture of wireless networks with mobile hosts.

2.1 Load Balancing in Homogeneous Broadcast Distributed Systems

The basic resources in any computing system, especially distributed and networked systems are the processors. But, these resources are not effectively and efficiently used [14]. For example, even though powerful processors of a system are not fully exploited, certain users of the system have to wait for long periods for completion of their background computational jobs [11]. Livny and Melman [13] calculated the probability that at least one job is waiting for service at one processor while at least one processor is idle. Their probability expression is given by: $P_{wi} = (1-P_0^N)(1-P_0^N-(1-P_0)^N)$ where, P_0^i is the probability that i processors are idle; P_0 is the probability that a processor is idle and N is the number of processors ($N>1$). The idle waiting condition indicates that it is possible to reduce the average job response time. To achieve improved performance, one needs to limit excessive communication among tasks and evenly distribute the workload among the computers. But, communication costs increases as processor utilization is uniform and tasks are evenly distributed in a system. Thus, the goals of limiting communication and balancing jobs are conflicting to one another, leading to tradeoffs in obtaining (near) optimal system performance [1].

2.2 Static vs. Dynamic Load Sharing

According to Nishikawa and Steenkiste [15], load sharing can be either dynamic or static, and the load sharing problem can be solved by statically or dynamically scheduling jobs among computers. In static load sharing, the task and data distribution are determined at compile time. Static load sharing is effective if the workload can be sufficiently well described before migration decisions are made. Rosing and Weaver [17] proposed that a compiler could do the task distribution possibly with input from the programmer. Job or task migration decisions are also made based on the assumptions about the jobs that are to be executed: execution time, current

load of the execution environment, transfer times and communication costs. But in a practical environment, this information is usually not known in advance and hence static load sharing is not ideal for systems where the workload fluctuates rapidly [11]. Also, another drawback with the static load sharing approach is that it uses only the information about the average system behavior but not the current state of the system. In dynamic load sharing, job migration decisions are based on the current state of the system, and hence rapid system state changes can be readily captured. Livny and Melman [13] proposed that if the P_{wi} in equation (1) can be reduced by transferring tasks from one processor to another, the expected turn around time for the tasks in the system would be reduced. Thus, if the current state of the system can be observed, then by maintaining a balanced load, performance improvement can be achieved.

2.3 Comparison of Process, Object and Program Migrations

Different forms of migration mechanisms could be employed in different situations to handle load sharing. Current distributed computing systems facilitate some forms of migration e.g., program, process or object migration, aiming to achieve better system performance. For example, the Galaxy [18] and the V4 [6] distributed systems support process migration. Mach [2] supports task migration and Emeralds [9] supports object migration.

Even though process and object migrations are attractive mechanisms for load sharing, the heterogeneity of a distributed system is increased with the joining and leaving of the mobile computers. Also, software facilities to support process and data migration do not adapt well to such heterogeneous dynamic environments [10]. On the other hand, program migration or job transfer mechanism offers the following facilities, which are attractive in load sharing [11] to:

- Collect statistical information about system workload
- Migrate programs written in any language
- Migrate both source and object code
- Invoke remote programs
- Migrate data

Therefore, we use job transfer mechanism for load sharing in this paper.

2.4 Load Sharing with Abstract Mobile Ticket Engine (AMTE)

Le et. al [10] proposed the use of an Abstract Mobile Ticket Engine (AMTE), in conjunction with program, data or object migrations, for load sharing in mobile computing environments. The AMTE is platform independent and can be easily added to any computer. The engine allows mobile computers to purchase unique abstract tickets (similar to a credit card) from a stationary host for use of foreign network resources. Mobile tickets can be purchased, used and reused facilitating the system and the mobile computer to work together on a use-based payment arrangement. The fields in the ticket include a globally unique ticket identification (combination of the unique IP number of the host which issues the ticket and a sequence number unique to that host), data and time of purchase and expiry, the place the ticket was issued, the place the ticket was last used, the intended destination (optional), the class, etc. The information maintained by the engine and stored in the ticket is sufficient for the system to keep track off mobile computers. Mobile computers are assumed registered at some stationary computer that can be reached by using its IP address.

The concept of AMTE can be extended to distributed and network systems that enable mobile computers to use the resources not provided by the mobile computer's home server. AMTE can be installed at the stationary base stations (MSSs) in a cellular network. More over, an AMTE can act as a client or a server depending on the type of service requested by the mobile user. A request

for service can be of different types: (1) buying an abstract mobile ticket for using foreign resources, (2) reconnecting with the network using the previously purchased ticket, (3) getting the results of the jobs submitted previously, (4) submitting jobs, etc.

In the above ticket model, the tickets were mainly used to track mobile computers and the AMTE handles the problem of load sharing and accessing foreign resources. Peer servers have an agreement on sharing of resources. We have enhanced the design of the tickets in order to achieve better load sharing and remote resource scheduling. The ticket design, in this paper, combines the features of a credit card and workload information submitted to a server.

2.5 Ticket Acquisition Models

Buttyan and Hubaux [5] discuss three different types of ticket acquisition models: **(i)** ***Outlet model:*** In this model, the user directly acquires the ticket from the service provider. The advantage of the outlet model is its simplicity, but it incurs considerable overhead in using complicated protocols to determine the optimal service provider that offers the best price and the best quality, or the combination of the two. **(ii)** ***Kiosk model:*** According to this model, a special service provider called the kiosk, sells the tickets for various service providers. Users do not have to contact each service provider if they need a ticket, but they can contact a kiosk, where information about the offers of several service providers is available. The disadvantage of the kiosk model is that it requires infrastructure (kiosk) to be built up and the users still need to decide themselves, which requires resources. **(iii)** ***Agency model:*** In the agency model, agencies buy tickets, for users, from the service providers. The advantage of this model is that users can obtain the best tickets in an easy way by allowing the agency to contact the service providers and find an optimal offer.
In this paper, the MSS acts as the kiosk and the clients can use a ticket for wide variety of services offered by different service providers. Hence, applying the above classification of tickets and ticket acquisition models, we can classify the tickets in this paper as of type t_4 and their acquisition model is a modified version of kiosk where the MSS or base stations of a cell itself acts as service providers.

2.6 History-driven Dynamic Load Sharing

Dynamic load sharing is likely to be suitable on systems where the workload fluctuates rapidly [12]. History-driven dynamic load sharing shows significant improvements [3] over conventional schemes, which assign jobs in a random, a fixed, or a worst order (worst ordering refers to an ordering in which the jobs are assigned to the busiest workstation first). But, it incurs a huge storage overhead in establishing a run-time history database at each MSS. Instead the tickets can be designed to maintain a small database pertaining to the applications of the MH. These tickets are passed along with the job in the mobile network. Since clients can submit the tickets at any MSS in the network, there is a distinct advantage in maintaining the run-time information at the tickets rather than at the MSS. Also, at run-time, application characteristics pre-dominate system characteristics, and identical processors are likely to be used through out the network, both of which support maintaining historical information in the tickets.

3. TICKET DESIGN

In this paper, we enhance the structure of tickets and propose their broader use, than in earlier works, for resource scheduling, load sharing and billing in mobile computing environments. A ticket has many fields, which contain information required for routing the results of computation

through the network and also for load sharing within a MSS. Figure 2 shows the different fields in the ticket, as designed in our previous work [19]. These fields are explained below:

- *Ticket Id*: The ticket id is unique to each ticket. It consists of four parts as shown in Figure 2. The first part identifies the SH whose identity is unique in the entire network, the second part identifies the MSS which is unique in a cell, the third part refers to the IP address of the mobile host in its home cell and the final part represents the serial number of the ticket.
- *Time of issue*: The time when the ticket was issued.
- *Expiration time*: The time when the validity of the ticket expires.
- *Account Balance*: When clients buy the ticket, the payment is credited to this field of the ticket. When they use computing resources, the executing MSS debits this field accordingly.
- *Application Id*: Each application submitted by the host, is given a unique identification.
- *Job Id*: A job is an instance of an application. A ticket may contain multiple instances (jobs) of an application. The MSS assigns a unique job id. The job id is used to identify the result at the destination station.
- *Transaction Id*: The MSS assigns a transaction id to the job for billing and communicating with the peer MSSs and the SH. The peer MSSs refer to the home MSS for billing using this transaction id. The transaction id assigned to a job at an MSS is unique. The transaction id is used to identify a job in the priority list during resource scheduling and load sharing any time at an MSS.
- *Job Status*: Job status could be *Active* (‘*A*’), *Suspended* (‘*S*’), *Waiting* (‘*W*’) or *Historical* (‘*H*’). Active jobs are those currently being run at the MSS. Each client may have more than one active job. Submitted jobs can also be suspended or waiting on a resource. They may become active later. Jobs that were run recently and which are likely to recur are tagged as historical. The MSS sets this field.
- *Destination*: The clients specify their intended cell or destination. The results of the computation will be routed to the MSS at the destination.
- *Output Data Expiration Date*: The client and the MSS agree upon an expiration date for the results. After the expiration date, the destination MSS may purge the results of execution in its database.
- *Deadline*: The deadline before which the MSS should finish executing a job. Some jobs e.g., weather predictions, may be time-critical. The client specifies the deadline.
- *Arrival Time*: The user can also specify the expected time of arrival at the destination. The results of the executed job should be available by this time.

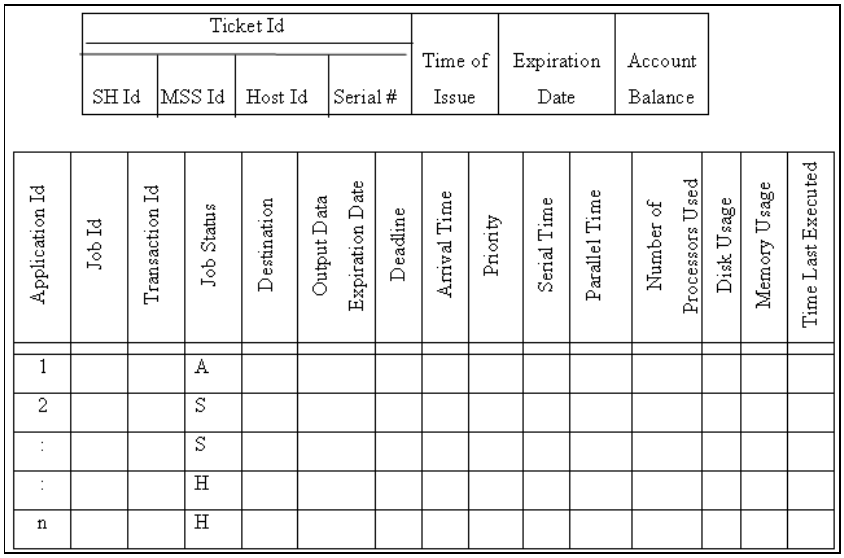

Figure 2: Ticket Structure (adapted from [19])

- *Priority*: The following priorities for the execution mode are assigned to jobs by the submitting client: Emergency, Urgent, Regular and Ordinary mode in descending priority. The billing rate for jobs with different priorities is different.
- *Serial Time*: The serial execution time of the job in its last run. The MSS updates this field. The client may also specify an expected value of serial time, particularly for the first run.
- *Parallel Time*: The parallel time of a job if it was run on a multiprocessor system. The MSS updates this field. The client may also specify an expected value of parallel time, particularly for the first run.
- *Number of Processors*: The number of processors used when the application was last run on a multiprocessor system. The MSS updates this field. The client may also specify an expected value for the number of processors, particularly for the first run.
- *Disk Usage*: This field represents the amount of disk space that the application used when it was last run or the expected disk usage. Either the MSS or the client updates this field. The client may specify an expected value of disk usage, particularly for the first run.
- *Memory Usage*: This field represents the amount of memory that the job used when it was last run or the expected memory usage. Either the MSS or the client updates this field. The client may specify an expected value of memory usage, particularly for the first run.
- *Time Last Executed*: This field tells the time at which the job was last executed. The results of the different applications in the ticket are purged using the Least Recently Used (LRU) algorithm, which uses this field.

4. LOAD SHARING, SCHEDULING AND BILLING

4.1 Cellular Mobile Network Architecture

In this section, we explain the cellular network architecture used in this work. We basically expand the cellular network architecture proposed in [4]. We had given a brief overview of the architecture in Section 1. The reader is suggested to refer to Figure 1 in Section 1 for further reading in this section.

We consider a four-layered cellular network architecture. The entire network is divided into multiple cells. Each cell is controlled by a mobile support station (MSS), which is similar to the base station in GSM networks. The mobile hosts (MHs) or clients constitute the bottom-most layer of the network. Each MH has a home cell and a home MSS with which it is registered. The MH is assigned an IP address unique in the home cell. The MH is also free to move from one cell to another with the support of "hand-shake" mechanism between the MSSs in the two cells. When an MH enters an alien cell, it registers with the MSS in the alien cell after the required identity validation. For simplicity, we assume the validation mechanism that is used in the MobileIP protocol. The MH is assigned a new foreign address that is unique in the alien cell. The MH can also use its home IP address to correspond with the MSS in its home cell. The MSS-MH link is wireless. All the MSSs are connected to each other and constitute the second layer of the network. An MSS is assigned a unique id in the network. We assume that the AMTE, described in section 2.4, can be installed at an MSS. MHs or clients can buy the tickets from the MSS. Clients can buy more than one ticket from an MSS and can also buy tickets from more than one MSS. We also assume that the MSS acts as the stationary host, providing the services requested (job execution, job forwarding, results storing) by the clients. Each MSS is assumed to be a shared memory multiprocessor system.

The supervisory hosts (SHs) constitute the third layer. The SH, similar to the base station controller in GSM networks, controls multiple cells. An MSS is connected to the SH that controls its cell. An SH is assigned a unique id in the network. The super supervisory host (SSH) constitutes the top-most layer in the network. All the SHs are connected to the SSH. The SSH regulates the operation of the entire network in co-ordination with the SHs. In this paper, we assume that the MSSs are the only stationary hosts that execute the jobs submitted by the clients. The SSH and SHs perform only load sharing and resource scheduling of the jobs, which is also performed by the MSSs in addition to job execution.

4.2 Load Sharing and Scheduling Algorithm

We propose a new load sharing and scheduling (LSS) algorithm that is to be executed at each MSS upon receiving a ticket from a client. The algorithm is shown in Figure 3 and explained in a step-wise fashion.

Procedure *LSS*

1. Check ticket validity.
2. If valid, assign *Job Id* and *Transaction Id*. Otherwise, set the above fields in the ticket to –1, implying invalid ticket and return the ticket to the client and exit.
3. The client uploads the job(s) to the ticket and sets the *Status* field of relevant jobs to either *Active*, '*A*'.
4. The MSS computes the net priority of the jobs submitted.

   $$Net\ Priority = \{(Priority)_{Origin} + (Priority)_{Mode}\} * Deadline$$

5. Schedule job based upon non-decreasing order of *Net Priority*.
6. If the MSS can meet the job's deadline, set the *Status* field on the ticket as *Active* and return a receipt to the ticket with the *Job Id* and *Transaction Id* assigned in Step 3. Otherwise, forward the ticket with the job to the SH and return a copy of the ticket to the client with the *Status* field set to *Suspended*, '*S*'. (Note: In forwarding, MSS merges small resource requests into blocks of larger requests).

end LSS

Figure 3: Load Sharing and Scheduling (LSS) Algorithm (adapted from [19])

Step 1: To execute a job or multiple jobs, a client first sends the ticket to the ticket engine at the MSS. Each MSS maintains a ticket database to keep track off the tickets bought, the balance amount on the ticket, the date of purchase and expiry of the ticket, etc. If the *MSS Id* in the *Ticket Id* field matches with that of the MSS (i.e., the client has bought the ticket from the MSS), the MSS checks for a valid entry for the *Ticket Id* field in its ticket database. If a match is found, the ticket is considered valid. If the *Ticket Id* field in the ticket refers to a different *MSS Id* (*Ticket Id* field includes the *MSS Id* from which the client buys the ticket), then the MSS contacts a peer MSS identified by the *MSS Id* in the *Ticket Id* field and requests for authentication. The peer MSS does a similar ticket database search as explained previously, and confirms whether it is a valid ticket or not.

Step 2: Depending upon the number of jobs requested, the MSS stamps the validated tickets with new job id(s) and transaction id(s) and returns the ticket to the client. Invalid tickets are returned stamped with a job id and transaction id of –1 and cannot be used for job submission.

Step 3: The client uploads the job(s) to the validated ticket. If there is no matching application for the job uploaded, the client treats the job as a new entry into the ticket and specifies the expected values for the *number of processors*, *serial time*, *parallel time*, *disk usage* and *memory usage* in appropriate fields in the ticket. The client marks the status field of the job(s) to be submitted as '*A*' (*Active Status*). The client specifies a destination MSS and the deadline before which the job should have been executed and the results available at the destination MSS. The client also specifies the *priority* for the execution mode of the jobs. Jobs can be executed in four different modes (in decreasing value of priorities): Ordinary, Regular, Urgent and Emergency modes. If the priority is not specified, jobs are by default executed in Ordinary mode. The client then submits the job(s) along with the ticket to the MSS.

Step 4: The MSS adds the information (all the fields relevant to the job in the ticket) about the newly submitted jobs to a *Job_Fields* database. The MSS computes the transmission delay that would be incurred in transferring the results of execution of the job to the destination MSS. The MSS updates the *deadline* field for the job in the *Job_Fields* database with the difference between the current *deadline* and the transmission delay. The MSS then computes the *net priority* of the submitted jobs. The *net priority* is given by: $Net\ Priority = \{(Priority)_{Origin} + (Priority)_{Mode}\} * Deadline$. The *priority due to origin* is computed using the *Ticket Id*. If the *MSS Id* in the *Ticket Id* field matches with that of the MSS (i.e., the client has bought the ticket from the MSS), then the client is treated as a home client and assigned a higher priority. If the MSS Id in the *Ticket Id* field does not match with that of the MSS (i.e., the client has not bought the ticket from the MSS), then the client is treated as an alien and assigned a lower priority. Figure 4 shows the numerical values of *priority due to origin* and *priority due to execution mode*. The net priority thus computed combines the features of the Earliest Deadline First (EDF) and Highest Priority First schemes and is suitable for soft real-time systems. Jobs with lower net priority value are invoked first.

Table 1: Priority Information in Tickets

Modes	Priority	Origin	Priority
Ordinary	4	Peer	2
Regular	3	Home	1
Urgent	2		
Emergency	1		

Step 5: The MSS maintains a sorted list of jobs (called *net_priority* list) based on increasing values of *net priority*. Each processor in the MSS is assumed to handle only one job at a time. The job with the lowest net priority is the first one to be assigned a processor for execution.

Step 6: If the MSS can meet the job's deadline, it sets the *Status* field on the ticket as *Active* and returns the ticket to the client with the *Job Id* and *Transaction Id* assigned in Step 3. The MSS immediately forwards the job to its SH if it cannot afford to complete the execution of the job before the updated delivery deadline or does not own sufficient resources (memory and processors) to execute the job. The *Status* field in the jobs forwarded to the SH is set to *Suspended*. If the MSS lacks resources, it may sometimes act as an agent by collecting all the jobs submitted and sending them to the SH for distribution to peer MSSs.

4.3 Rescheduling

The MSS maintains a sorted list of jobs (called *job_execution* list) that are currently executing, in the order of their earliest finish time. A job once scheduled at an MSS is not pre-emptable. Once the resources are exhausted, the MSS attempts to do a rescheduling process before sending the unexamined jobs (jobs that are not yet considered for scheduling) in the *net priority* list to the SH. If after the completion of a job or jobs in the execution list, an unexamined job in the *net_priority* list can be still executed and its delivery deadline guaranteed, then that job is designated to be in the *Waiting* (‘*W*’) state and is not forwarded to the SH. The job is scheduled to execute once the currently running job(s) (with which the former is compared during rescheduling) finish execution. Jobs that cannot be executed at the MSS even after rescheduling, are immediately forwarded to the SH. A significant feature of our resource scheduling algorithm is that the MSS at which the job is submitted is immediately able to determine whether it can handle the job; otherwise it forwards the job to the SH. In other words, Steps 5 and 6 of *LSS* and rescheduling can be executed without any appreciable delay (by a simple “dequeue” operation into the *net_priority* and *job_execution* lists) and thus the deadline guarantee of the jobs not affected.

Rescheduling comes with a drawback. Since the jobs selected for execution after rescheduling, have a higher wait time, the turn around time of the jobs also increases. But rescheduling guarantees that the deadline of the job can be met and excessive transmission delays due to job forwarding can be avoided. If after the initial scheduling (steps 5 and 6 of LSS), all the unscheduled jobs were forwarded to the SH, then there may be a chance that certain jobs are executed beyond their deadline at the peer MSSs. We see this as a trade off between the turn around time and the deadline guarantee.

4.4 Scheduling and Load Sharing at the SH

The SH has “read” access to information on the availability of resources (memory and processors) at its constituent MSSs. The SH maintains a sorted list (called the *MSS_Resource* list) in increasing order of the amount of free resources (number of free processors and unused memory) available at each of its constituent MSSs. The SH also maintains a sorted list (called the *SH_net_priority* list) based on the *net priority* of the jobs sent by its constituent MSSs. The SH schedules the jobs in the *SH_net_priority* list to the MSS, which can finish the job at the earliest and can guarantee the delivery deadline of the job at the destination MSS. The delivery deadline of the job varies with different MSSs since the transmission delay in the SH-MSS-destination MSS links vary. If there exists more than one candidate MSS, which can provide the deadline guarantee and the same earliest finish time, the job is assigned to the MSS that has relatively higher resources than the other candidate MSSs. If none of the MSSs are able to provide the deadline guarantee, the job is forwarded to the SSH.

4.5 Scheduling and Load Sharing at the SSH

The SSH has "read" access to the information on resource availability of all the MSSs in the network. A scheduling procedure similar to that described for SH is adopted. But in this case, even if there exists no candidate MSS satisfying the deadline guarantee, the job is scheduled to the MSS that can complete the job at the earliest. This approach will suit for soft real-time applications and also for hard real-time applications with slight flexibility in deadline.

4.6 Billing

The home MSS bills the mobile clients for the workload handled using the *Account Balance* field of the tickets. Higher bills incur for jobs with higher priorities. Peer MSSs communicate among themselves to consolidate financial transactions using *transaction id*s and *job id*s. MSSs that execute and forward the results to the destination, debit certain amount from the *Account Balance* field of the ticket.

In a homogeneous system where all the MSSs possess identical resources to handle jobs, the ticket-based job forwarding and load sharing model proposed in this paper distributes the load evenly in the network irrespective of the MSS at which the jobs are submitted. If an MSS is unable to execute and finish a job before the deadline, it immediately forwards the job to its SH, which distributes the job to peer MSSs. If the SH is unable to find an MSS that could execute and finish the job before the deadline, it forwards the job to the SSH. The SSH has a global view of resource availability in the network and finds a suitable MSS to execute the job and at the same time maintaining the deadline guarantee. Thus, the ticket model distributes the workload evenly leading to equal earnings for all the MSSs.

In a heterogeneous system, MSSs vary in their resource content. The ticket-based job forwarding and scheduling model is equally well applicable for heterogeneous systems. The MSSs are loaded relative to their potential of handling the jobs and an MSS is not overloaded with jobs that it cannot handle. Thus, the ticket model distributes the jobs in the network depending on the capacity of an MSS and hence leads to earnings proportional to the availability of resources at an MSS.

5. SIMULATIONS

Both the ticket and FIFO models are subjected to the same set of experimental data at simulation conditions explained below in Sections 5.1 through 5.4. For each set of maximum deadline condition, the runs are made for a T-1 link and for an OC-3 link and in each case with 5, 25 and 50 processors at each MSS. The percentage deadline guarantee and the turn around time for the jobs in the ticket model and the FIFO model are measured. The percentage of jobs in the ticket model whose turn around time is less than that in the FIFO model is then calculated.

5.1 Simulation Environment

We simulated the four-layered micro-cellular network architecture using a mobile test bed consisting of cells adjacent to each other, with one MSS controlling a cell. MHs are registered as home clients to an MSS. An MH can be a home client in more than one cell. The MSS – SH – SSH, MSS – MSS links are assumed connected as in conventional wired networks. The MSS – MH link is wireless. Free space propagation channel is assumed. The MHs roam in all directions at a predefined average speed and a reflecting boundary model is assumed. The jobs and the results of execution are forwarded respectively to the target MSS (the MSS that could execute the job) and destination MSS via the MSS – SH – SSH – SH – MSS route.

The MSSs are assumed to be shared memory multiprocessor systems. The experiments are simulated in both homogeneous and heterogeneous environments. In the case of a heterogeneous environment, the number and speed of processors at each MSS varies and is less than a fixed maximum value (5, 25, 50). The total shared memory at each MSS varies from 500MB to 5GB. The speed of the processors at the MSS is varied using a speed factor whose value ranges from 0.01 to 1. It is assumed that the execution time of jobs listed in the tickets is based on a speed factor of 1. The execution time of a job at an MSS is calculated as the execution time of the job as listed in the ticket divided by the Speed factor of the processors at the MSS.

In the case of a homogeneous environment, all the MSSs use the same fixed number of processors (5, 25, 50) with identical speed factor. The total shared memory at each MSS is fixed (2GB). The size of the unexecuted applications and that of the results of the applications are each less than or equal to 100MB. The execution time of the applications is less than or equal to 5 hours.

In addition to simulating the ticket-based scheduling and load sharing model, whose operation was explained in the previous sections, we also simulated a First-In First-Out (FIFO) based scheduling model for the submitted jobs at each MSS. In the FIFO model, the MSS never forwards the jobs for execution to any other peer MSS. Regardless of the priority and deadline of the jobs, the MSS executes the jobs on a first-come first-serve basis.

5.2 Simulation Parameters and Performance Measurement

We study the performance of the ticket model with respect to the percentage deadline guarantee and percentage of jobs with lower turn around time when compared to that in a FIFO model. The percentage deadline guarantee is percentage of the submitted jobs whose results of execution are delivered to the destination MSS before their deadline. The turn around time is the difference between the delivery time at the destination MSS and the initial submission time at an MSS. The performance of the two models is studied by varying the following parameters:

(1) *Client arrival rate:* We assume the client arrival rate into the network is in a Poisson distribution. Each client can submit more than one job (maximum 6 jobs per ticket) at an MSS. The mean arrival rate of the clients is varied from 10 to 90 clients per hour over a time period of 24 hours (a day).

(2) *Link speed:* The speed of the wired links is 1.544Mbps (for T-1 link) and 155Mbps (for OC-3 link).

(3) *Number of processors:* In the case of a heterogeneous system, the maximum number of processors at any MSS used is 5, 25 and 50. For example, if the maximum number of processors at an MSS is fixed as 5, then the number of processors at an MSS can vary randomly from 0 to 5. In the case of a homogeneous system, the number of processors is the same at all the MSSs and the values are 5, 25 and 50.

(4) *Shared memory:* All the MSSs are assumed to be shared memory multiprocessor systems. In the case of a heterogeneous system, the memory at the MSS is varied from 500MB to 5GB. For example, one MSS in the system can possess 1GB of memory while another system can possess 3.2GB of memory. In the case of a homogeneous system, all the MSSs are assumed to possess 2GB of memory. This could be any value but we chose 2GB since it is approximately the average of the range of memory used in the heterogeneous system considered.

(5) *Maximum deadline:* All jobs submitted in the network are associated with a deadline for completion. The simulation is run for 4, 6 and 9 hours of maximum deadline of the jobs submitted.

5.3 WAN Switching and Link Speeds

There are mainly two types of switching in wide area network (WAN) links: circuit switching and packet switching [16]. Circuit switching is ideal when data must be transmitted quickly, must arrive in sequencing order and at a constant arrival rate. Thus, circuit switching networks are used while transmitting real-time data such as audio and video. Packet switching networks are more efficient and robust for data that is bursty in its nature and can tolerate some amount of delay. Since the performance of our ticket model is measured with reference to the percentage deadline guarantee and percentage of jobs with lower turn around time than a FIFO model, we consider the MSS-MSS, MSS-SH, SH-SSH links as circuit switched.

We simulate the MSS-MSS links as T-1 links and OC-3 links. T-1 links [7] are the most commonly used digital lines in the US, Canada and Japan. T-1 links use copper wires and span distances within and between major metropolitan areas. The links carry 24 pulse code modulation (PCM) signals using time division multiplexing (TDM) at an overall rate of 1.544 million bits per second (Mbps). OC-3 links [7] are a type of Synchronous Optical Network (SONET) links that transmit digital signals on optical fiber. Optical fiber carries much more information than conventional copper wire and is not subject to electromagnetic interference and signal retransmission. OC-3 links operate at a data rate of 155Mbps, almost 100 times the speed of T-1 links. We study the performance of the ticket and FIFO models with respect to these two links having wide difference in their bandwidths.

5.4 Delay Calculations

The delay in a circuit switched network is the sum of the link propagation delay, the serialization delay, the transmission delay and the queuing delay [8]. We assume that the MSS-SH and MSS-MSS links under the control of a particular SH are pre-established. Hence, the connection set up cost is constant and need not be considered in delay calculations. The link propagation delay is the time a transmitted bit needs to travel from one end of a link to the other end. It is dependent only on the speed at which signals travel on the transmission medium and the length of the link. Speed of light is $3*10^8$ m/sec and the speed of signals in most guided media is $2*10^8$ m/sec [8]. Hence, even for a transmission between nodes that are $2*10^5$ km apart, the link propagation delay is only a second, which we assume does not significantly affect the deadline guarantee and the percentage of jobs with lower turn around time. The queuing delay is dependent on the length of the queues at the routers. Even though the queuing delay is a very dynamic quantity, we have no resources to compute the queuing delay at the routers. Hence, we assume that the network is not loaded very heavily which would otherwise cause a significant queuing delay at the routers. Transmission delay is the time it takes to transmit all the bits of a packet onto a link. The transmission delay is only dependent on the transmission size and the transmission rate. In circuit-switched networks, the transmission delay is experienced only at the first node and not at intermediate nodes. The serialization delay or the insertion delay is the amount of time it takes to place the bits of a packet on the transmission wire. The serialization delay is a function of the size of the packet and the speed of the port. We assume the packet size to be the maximum transmission unit (MTU) size (1500 bytes) used in conventional networks. The serialization delay

for a T-1 link is 5µs per byte while that of an OC-3 link is 0.05µs per byte. We assume the maximum size of the application or the results of the application to be 100MB. Hence, the serialization delay is appreciable and is experienced at each hop. Considering all the above said features of the delays, we compute the delay in the network to be the sum of the transmission and serialization delays.

5.5 Results for Homogeneous Systems

From the simulation results for homogeneous systems, it can be inferred that for a network with 5 processors at each MSS (i.e., limited resources), the percentage of jobs with a lower turn around time (in the ticket model than in the FIFO model) decreases as the network traffic increases. But for a network with 25 and 50 processors at each MSS (i.e., sufficient and/or excessive resources), the percentage of jobs with a lower turn around time does not vary significantly with the network traffic. One can also observe that in most of the results, an OC-3 link yields a slightly higher percentage of jobs with lower turn around time when compared to that of a T-1 link. Similar to the case of heterogeneous systems, it can be inferred from the results that the percentage deadline guarantee in the ticket model decreases as the network traffic increases and for a fixed client arrival rate, increases as the maximum number of processors at an MSS increases, the latter being as expected. The parameters representing the vertical bars in Figures 4 through 9 represent the link bandwidth (T-1 or OC-3, in Mbps) and the number of processors at each MSS.

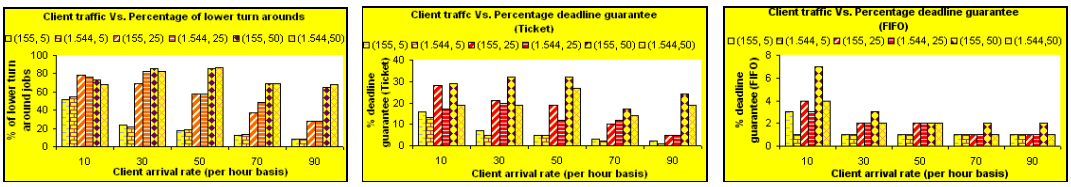

Figure 4: Simulation Results for Homogeneous Systems (Maximum Deadline: 4 Hours)

Similarly, the type of link (T-1 or OC-3) does seem to cause variation in the percentage deadline guarantee. In most of the results observed, an OC-3 link provides better deadline guarantee than that by a T-1 link. This is attributed to higher serialization and transmission delays for a message in a T-1 link when compared to that in an OC-3 link. Similar to the case of heterogeneous systems, in the case of a FIFO model, it can be inferred that the percentage deadline guarantee decreases as the network traffic increases, which is also observed in the ticket model. It can also be observed that the type of link (T-1 or OC-3) has a significant effect on the percentage deadline guarantee for the FIFO model. As expected, an OC-3 link yields a higher percentage deadline guarantee than that of a T-1 link.

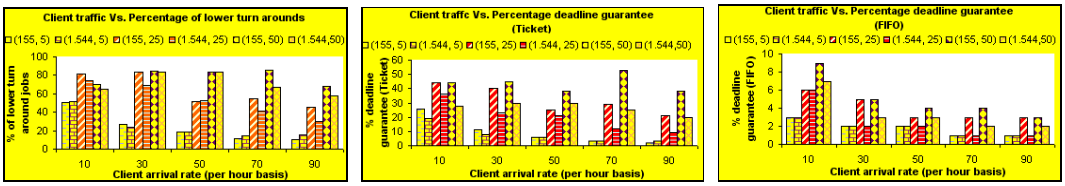

Figure 5: Simulation Results for Homogeneous Systems (Maximum Deadline: 6 Hours)

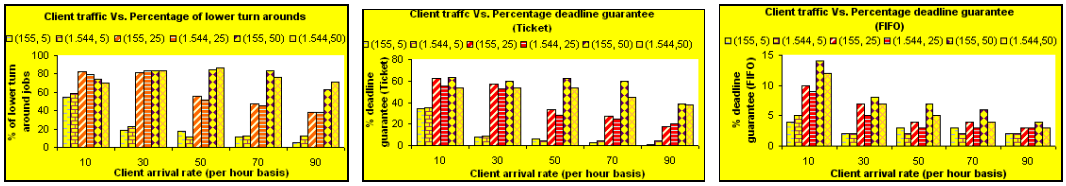

Figure 6: Simulation Results for Homogeneous Systems (Maximum Deadline: 9 Hours)

5.6 Results for Homogeneous Systems

From the simulation results for heterogeneous systems, it can be inferred that the percentage of jobs with a lower turn around time (in the ticket model than in the FIFO model) decreases as the network traffic (related to the client arrival rate) increases and for a fixed client arrival rate, increases as the maximum number of processors at an MSS increases, the latter being as expected. Also, it is interesting to observe that in majority of the runs, the type of link (T-1 or OC-3) does not significantly cause variation in the percentage of jobs with a lower turn around time. Similarly, it can be inferred from the results that the percentage deadline guarantee in the ticket model decreases as the network traffic increases and for a fixed client arrival rate, increases as the maximum number of processors at an MSS increases, the latter also being as expected. The type of link (T-1 or OC-3), as expected, does seem to cause variation in the percentage deadline guarantee. In most of the results observed, an OC-3 link provides better deadline guarantee than that by a T-1 link. This is attributed to higher serialization and transmission delays for a message in a T-1 link when compared to that in an OC-3 link.

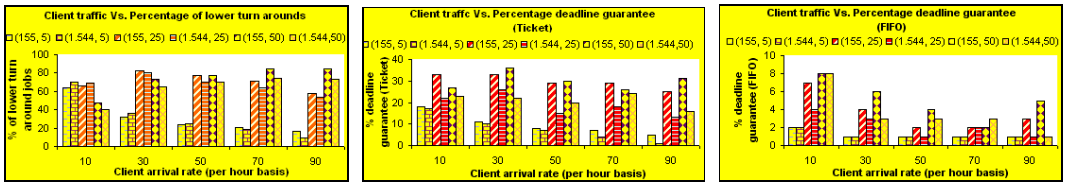

Figure 7: Simulation Results for Heterogeneous Systems (Maximum Deadline: 4 Hours)

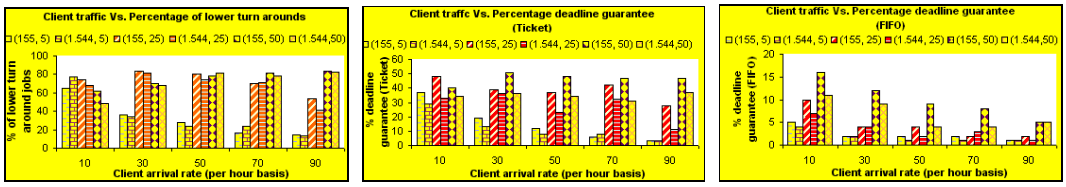

Figure 8: Simulation Results for Heterogeneous Systems (Maximum Deadline: 6 Hours)

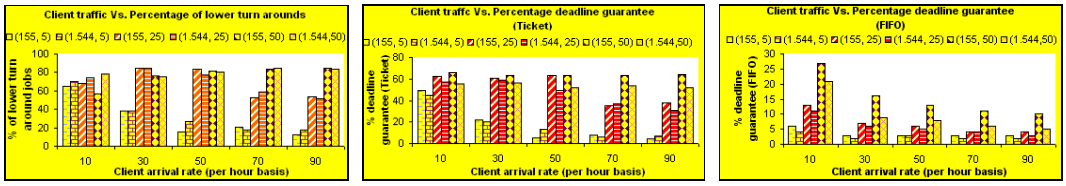

Figure 9: Simulation Results for Heterogeneous Systems (Maximum Deadline: 9 Hours)

In the case where the maximum deadline of the applications is 9 hours, while the maximum execution time of applications is 5 hours, one can observe only a marginal increase in the

percentage deadline guarantee when compared to the results for the other two deadlines. In the case of a FIFO model, it can be inferred that the percentage deadline guarantee decreases as the network traffic increases, which is similar that observed in the ticket model. Also, one can observe that at lower client arrival rates, an increase in the number of processors increases the percentage deadline guarantee (but not significantly when compared to that of a ticket model).

6. CONCLUSIONS AND FUTURE WORK

For both homogeneous and heterogeneous systems, comparing the performance of both the ticket and the FIFO models, one can observe that the percentage deadline guarantee in the ticket model is higher than that in the FIFO model for all the combinations of the simulation parameters. And as expected, higher the resource availability, higher the percentage deadline guarantee in the ticket model than that in the FIFO model. We have assumed that the jobs executed at the MSS are independent. In other words, the execution of a job once scheduled at an MSS is not dependent on any other job currently being executed or waiting to be executed in other MSSs or is not dependent on the resources of any other MSS. Future work should address inter-application dependencies. Load sharing among the peer MSSs can be performed based on different objective functions such as reduced resource usage, reduced transfer cost, etc. without affecting the priorities and the deadline requirements contained in the ticket. Also, a job once assigned to an MSS is assumed to be non-preemptable prior to completion. Future work should be on systems in which jobs once assigned to an MSS for execution can be preempted prior to completion. In other words, context switching must be considered.

7. REFERENCES

[1] K. M. Baumgartner and B. W. Wah, “A Global Load Balancing Strategy for a Distributed System,” Proceedings of IEEE Workshop on Future Trends in Distributed Computer Systems in the 90’s, pp.14-16, 1988.

[2] D. Black, “Scheduling Support for Concurrency and Parallelism in the Mach Operating System,” IEEE Computer, v.23, n.5, pp.35-43, 1990.

[3] M. Bozyigit, “History-driven Dynamic Load Sharing for Recurring Applications on Networks of Workstations,” Journal of Systems & Software, v.51, no.1, pp. 61-72, 2000.

[4] K. Brown and S. Singh, “Network Architecture for Mobile Computing,” Proceedings-IEEE INFOCOM, v.3, pp.1388-1396, 1996.

[5] L. Buttyan and J-P. Hubaux, “Accountable Anonymous Service Usage in Mobile Communication Systems,” EPFL SSC Technical Report No. SSC/1999/016, 1999.

[6] V. R. Cheriton, “The V Distributed System,” Communications of the ACM, v.13, no.3, pp.314-333, 1988.

[7] A. Croll and E. Packman, “Managing Bandwidth:: Deploying Across Enterprise Networks,” Prentice Hall, 2000.

[8] G. Huston, “Internet Performance Survival Guide: QoS Strategies for Multiservice Network,” John Wiley & Sons, 2000.

[9] E. Jul and N. Hutchinson, “Grained Mobility in The Emerald System,” Transactions on Computer Systems, v.6, no.1, pp.128-133, 1988.

[10] P. D. Le and B. Srinivasan, “A Migration Tool to Support Resource and Load Sharing in Heterogeneous Computing Environments,” Computer Communications Journal, pp.361-375, 1997.

[11] P. D. Le, V. Malhotra, N. Mani and B. Srinivasan, “Resource and Load Sharing in Mobile Computing Environments,” Proceedings IEEE Region 10th Annual Interantional Conference,v.1, pp.82-85, 1999.

[12] H. C. Lin and C. S. Raghavendra, “A Dynamic Load Sharing Policy with a Central Job Dispatcher,” IEEE Transactions on Software Engineering, v.18, no.2, pp.149-158, 1992.

[13] M. Livny and M. Melman, “Load Balancing in Homogeneous Broadcast Distributed Systems,” Proceedings of the ACM Computer Network Performance Symposium, v.11, no.1, pp.47-55, 1982.

[14] M. W. Mutka and M. Livny, "Scheduling Remote Processing Capacity in a Workstations-Processor Bank Computer System," Proceedings 7th International IEEE Conference on Distributed Computing Systems, pp. 47-54, 1993.
[15] H. Nishikawa and P. Steenkiste, "A General Architecture for Load Balancing in a Distributed-Memory Environment," Proceedings of the 13th IEEE International Conference on Distributed Computing Systems, pp.47-54, 1993.
[16] R. Perlman, "Interconnections: Bridges, Routers, Switches, and Internetworking Protocols," 2nd edition, Addison-Wesley Publishing Co., 1999.
[17] M. Rosing and R. P. Weaver, "Mapping Data to Processors in Distributed Memory Computations," Proceedings of the 5th IEEE Distributed Memory Computing Conference, pp.884-893, 1990.
[18] B. Vickers and B. R. Badrinath, "A Generalizable Service Architecture for Mobile Networks," International Workshop on MoMuc, pp.221-225, 1999.
[19] S. Baskiyar and N. Meghanathan, "Scheduling and Load Sharing in Mobile Computing using Tickets," Proceedings of the ACM-Southeast Conference, pp.175-179, 2001.