Baskiyar, S. and Meghanathan, N., "Scheduling and load balancing in mobile computing using tickets," Proc. 39th SE-ACM Conference, Athens, GA, 2001.

# SCHEDULING AND LOAD SHARING IN MOBILE COMPUTING USING TICKETS

*Sanjeev Baskiyar, Ph.D. and Natarajan Meghanathan*
*Department of Computer Science and Software Engineering*
*Auburn University, Auburn, AL 36830*
*e-mail: baskiyar@eng.auburn.edu*

**Abstract**

Load sharing in mobile computing environments is challenged by frequent network disconnections, widely varying bandwidths among wired and wireless links, limited computing power of Mobile Hosts (MHs) and transient servers due to frequent hand-off. We consider a three-layered network architecture consisting of Mobile Hosts (MHs), Mobile Support Stations (MSSs) and Supervisory Hosts (SHs). We outline the design of tickets, which combines the features of a credit card and workload information submitted to a server. Clients submit their database and computation-intensive applications to a MSS using tickets. This paper proposes a broader use of tickets, than in earlier works, in scheduling, load sharing and billing in mobile computing environments. A ticket contains valuable information to route the results of computation through the network, to schedule and load balance jobs in the network using priority and historical run-time information of jobs executed at the MSS. We outline a load sharing and scheduling algorithm to be executed at an MSS upon receiving a job with a ticket from the client. Job transfer rather than migrating server objects has been used since it avoids the context-switching overhead on the secondary host.

## 1 Introduction

In a mobile network having a micro-cellular architecture, a SH manages a group of MSSs, also called base stations, with each cell having a MSS as shown in Figure 1. Communication from one cell to another goes via the SH. Mobile clients may need to upload database and computation intensive applications to a MSS. An application (e.g., internet search) could be run by more than one client. A job refers to an invocation of one of these applications. The applications at a SH or MSS may be from a client or a peer MSS. In order to identify the

clients and to aid in scheduling, load sharing and billing, we propose that a MSS issue a ticket to a client.

The organization of this paper is as follows. In Section 2, we summarize the past work in load sharing in mobile computing using tickets. The design of the tickets is presented in Section 3. In Section 4, we address the functionality of the MSS vis-à-vis handling the tickets and jobs and also describe the mobile network architecture used for ticket transaction. Also, an algorithm for scheduling and load sharing jobs based on tickets and a mechanism of billing the clients using tickets has been discussed in Section 4. In Section 5, we present conclusions and suggestions for future work.

## 2 Background

In order to support high data rates across networks, a micro-cellular architecture [5,7] has been proposed. Kevin and Singh [10] proposed a three level hierarchical architecture for wireless networking in mobile computing. At the lowest layer is the MH, as shown in Figure 1, which communicates with the MSS node in each cell. The SH controls several MSSs. The SH is connected to the fixed wired Internet and handles routing and protocol details. All connections set up by a MH or MSS pass through their SH. The SH maintains an up-to-date database of the availability of the MSSs belonging to it. The SH has also access to such a database of peer SHs. We consider a micro-cellular architecture, as described above, in this paper.

Mobile clients often need to execute applications in a heterogeneous computing environment. Although the use of network protocols to allow foreign network resources to be easily discovered and accessed by mobile clients has been investigated [1,8], resource and load sharing in mobile computing environments is one of the main problems [6] and an active research area. Le. et.al [11] have proposed a method for load sharing on mobile computing environments, using an Abstract Mobile Ticket Engine (AMTE) in conjunction with program, object and data migrations. The AMTE facilitates the mobile computers to purchase unique abstract tickets from a stationary host for later use of computing resources. The AMTE could be installed in the MSS. In their 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 balancing and remote resource sharing. The ticket design, in this paper, combines the features of a credit card and workload information submitted to a server.

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., data, process, or object migration, aiming to achieve better system performance. For example, the Galaxy [15] and the V [4] Distributed Systems support process migration. Mach supports task migration [2] and Emerald supports object migration [9]. The process and object migrations are attractive mechanisms for load sharing. However, since the heterogeneity of a distributed system is increased with the joining and leaving of mobile computers, the aforementioned mechanisms are expensive. Also, software facilities to support process and object migrations do not adapt well to such heterogeneous dynamic environments [11]. On the other hand, program migration or job transfer mechanism offers the following facilities, which are attractive in load sharing [12]:

- to collect statistical information about system workload
- to migrate programs written in any language
- to migrate both source and object code
- to invoke remote programs
- to migrate data

Therefore, we use job transfer mechanism for load sharing.

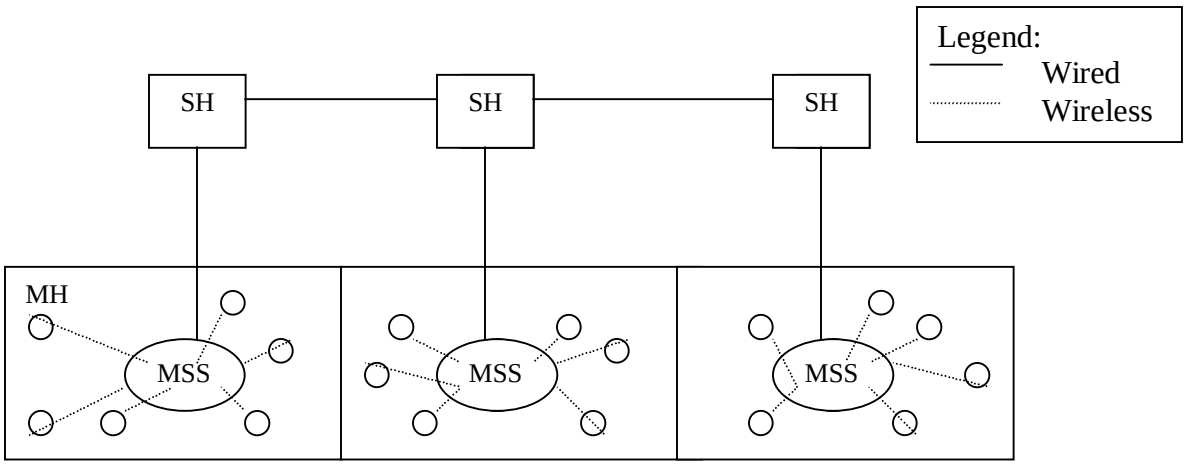


**Figure 1**. Three-Layered Mobile Network Architecture

In this paper, we address issues in history-driven dynamic load sharing. Dynamic load sharing is likely to be suitable on systems where the workload fluctuates rapidly [13]. 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). However, establishing a run-time history database at each MSS incurs storage overhead. Instead, the tickets could be used to maintain a small database pertaining to the applications of the ticket-owner or the mobile host. These tickets will be passed along with the job in the mobile network. Since the clients may submit the tickets at any MSS in the network, there is a distinct advantage in maintaining the run-time information of the applications in the ticket rather than at the MSS. Furthermore, at run-time application characteristics predominate system characteristics [3] and identical processors are likely to be used throughout the network [14], both of which support maintaining historical information in the tickets.

## 3 Design of the Ticket

The tickets have many fields, which contain information required for routing the results of computation through the network and also load sharing within a MSS. Figure 2 shows the different fields in the proposed ticket. 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.

- *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.

| Ticket Id | | | | Time of Issue | Expiration Date | Account Balance | Transaction Id |
|---|---|---|---|---|---|---|---|
| SH Id | MSS Id | Host Id | Serial # | | | | |

| Application Id | Job Id | Job Status | Destination | Output Data Expiration Date | Deadline | Arrival Time | Priority | Serial Time | Parallel Time | Number of Processors Used | Disk Usage | Memory Usage | Time Last Executed |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | A | | | | | | | | | | | |
| 2 | | S | | | | | | | | | | | |
| : | | S | | | | | | | | | | | |
| : | | H | | | | | | | | | | | |
| n | | H | | | | | | | | | | | |

**Figure 2**. Proposed Ticket Structure

- *Application Id*: Each application submitted by the host, is given a unique identification.
- *Job Id*: The MSS assigns a unique job id. This job id is used to identify the result at the destination station.
- *Job Status*: Job status could be Active (A), Suspended (S) or Historical (H). Active applications are those that are currently being run at the MSS. Each client may have more than one active application. Submitted applications can also be suspended, waiting on a resource. They may become active later. Applications, which were run recently and which are likely to recur are tagged as historical. The MSS sets this field.

- *Destination*: The clients specify their intended destination. The results of the computation will be routed to that 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.
- *Priority*: The following priorities 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 application 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 application was last executed. The results of the different applications in the ticket are purged using the LRU algorithm, which uses this field.

## 4 Load Balancing, Scheduling and Billing at MSS

To execute a job, the mobile host first sends a ticket, to the ticket engine at the MSS. The MSS validates the ticket's identity and stamps the ticket with a *job id* and a *transaction id*. The ticket is then returned to the client. The client sets the *status* field on the ticket to *Active* for the relevant job and submits it along with the ticket. The home MSS, at which the ticket is submitted, schedules the application using a priority scheme.

The MSS maintains a count of the tickets, the account balance on the tickets, it has issued so far, and a statistical information of the job traffic that it receives during fixed intervals of the day, all of which it uses to estimate the expected number of tasks that will arrive during any interval of time. The above information is useful to the MSS in load sharing and balancing. If a MSS lacks resources to handle jobs, it will act as an agent, collecting tickets and jobs from clients and forwarding them to its SH. The SH in turn assigns jobs to peer MSSs or a peer SH based on availability. A MSS may also assemble tickets, each of which may be for a small job, into requests of larger blocks of computing resources. Most likely computing resources may be less expensive to buy in blocks rather than in small portions.

Even though, multiple jobs may be submitted using a single common ticket, all of them, need not be executed at a single MSS. An application-level scheduling algorithm based on the information contained in the tickets is proposed. The jobs that are submitted at a MSS may originate from clients and peer MSSs with differing priorities. As mentioned in the *priority* field description of the ticket, jobs may be in Emergency, Urgent, Regular and Ordinary modes.

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

**Figure 3**. Priority information in tickets

Figure 3 shows the priority assignment. The net priority is computed as follows:

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

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. Applications with lower net priority value are invoked first. The Load Sharing and Scheduling (LSS) algorithm shown in Figure 4, explains the sequence of actions taking place at the MSS upon receiving a ticket.

---

**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. Compute

   $$Net\ Priority = \{(Priority)_{Origin} + (Priority)_{Mode}\}^* Deadline$$
4. Schedule job based upon non-decreasing order of *Net Priority*.
5. If the MSS can meet the job's deadline, set the *Status* field on the ticket as *Active* and return it to the client 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 4**. Load Sharing and Scheduling (LSS) Algorithm

If the MSS owns a Network of Workstations (NOW), it executes a load-balancing algorithm in order to distribute the constituent tasks of the job onto different workstations. The run-time history of the application contained in the tickets will be useful in reducing the overhead associated with parallelizing [3].

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.

## 5 Conclusion

Since mobile computer environments are heterogeneous as well as dynamic, it is difficult to perform load sharing in such environments than in distributed systems. This paper is the first to detail the design of the tickets and their use in scheduling, load sharing, and billing among peer MSS in a three-layered network architecture.

We have assumed that the applications submitted at the MSSs are independent. Future work must address inter-application dependencies. Also, load sharing at a home MSS could be performed based on different objective functions such as minimizing billing costs, resource usages, cost in transferring results, while satisfying the priority requirements contained in the ticket.

## References


[1] P. Bhagawat and C. Perkins, "Mobile Networking based on Internet Protocol," *IEEE Personal Communication Magazine*, 1(1), Feb 1994.

[2] D. Black, "Scheduling Support for Concurrency and Parallelism in the Mach Operating System," *IEEE Computer*, vol.23, no. 5, pp. 35-43, May 1990.

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

[4] V.R. Cheriton, "The V Distributed System," *Communications of the ACM*, vol. 13, no. 3, pp.314-333, Mar. 1988.

[5] Duchamp, Feiner and Maguire, "Software Technology for Wireless Mobile Computing," *IEEE Network Mag.*, pp. 12-18, Nov. 1991.

[6] Forman and Zahorjan, "The Challenges of Mobile Computing," *IEEE Computer*, 27(4), pp. 38-47, Apr. 1994.

[7] Goodman, "Trends in Cellular and Cordless Communications," *IEEE Communications Magazine*, pp. 31-40, June 1991.

[8] Ioannidis and Maguire. Jr, "G.Q based protocols for Mobile Internetworking," *Proceedings of ACM SIGCOMM*, pp. 235-24, 1991.

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

[10] Kevin and Singh, "Network Architecture for Mobile Computing," *Proceedings-IEEE INFOCOM*, vol. 3 IEEE, Piscataway, NJ, pp. 1388-1396, 1996.

[11] Le, Malhotra , Mani and Srinivasan, "Resource and Load Sharing in Mobile Computing Environments," IEEE Region 10 Annual Interantional Conference, Proceedings/TENCON. vol. 1, IEEE, Piscataway, NJ, pp. 82-85, 1999.

[12] Le and Srinivasan, "A Migration Tool to Support Resource and Load Sharing in Heterogeneous Computing Environments," *Computer Communications Journal*, Elsevier, U.K, pp. 361-375, 1997.

[13] H.C. Lin and C.S Raghavendra, "A Dynamic Load Sharing Policy with a Central Job Dispatcher," *IEEE Transactions on Software Engineering*, vol. 18, no. 2, pp. 149-158, Feb. 1992.

[14] S.H.Russ., Meyers, Rajagopalan, Rajan, Reece, Robinson, and Tan, "Hector-An Agent-Based Architecture for Dynamic Resource Management," *IEEE Concurrency*, April-June 1999.

[15] P.Sinha, H Ashishara, K.P. Birman, X. Jia, M. Maekawa and K. Shimizu, "The Galaxy Distributed Operating System," *IEEE Computer*, pp. 34-41, Aug. 1991.