Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

LOAD SHARING IN HETEROGENEOUS DISTRIBUTED SYSTEMS

Helen D. Karatza

Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, GREECE

Ralph C. Hilzer

Computer Science Department
California State University, Chico
Chico, CA 95929-0410, U.S.A.

ABSTRACT

Load sharing is key to the efficient operation of distributed systems. This paper investigates load sharing policies in a heterogeneous distributed system, where half of the total processors have double the speed of the others. Processor performance is examined and compared under a variety of workloads. Two job classes are considered. Programs of the first class are dedicated to fast processors, while second class programs are generic in the sense that they can be allocated to any processor. The objective is to find a policy that results in good overall performance while maintaining the fairness of individual job classes. Simulation results indicate that the performance of the best method depends on system load.

1 INTRODUCTION

Distributed systems provide attractive scalability in terms of computation power and memory size. Generally, in distributed systems no processor should remain idle while others are overloaded. It is preferable that the workload be uniformly distributed over all of the processors. It is important to efficiently utilize computational power, especially when load distribution is necessary.

The purpose of load sharing algorithms is to ensure that no processor remains idle when there are other heavily loaded processors in the system. With sender-initiated algorithms, load-distribution activity is initiated when an over-loaded node (sender) tries to send a task to another under-loaded node (receiver). In receiver-initiated algorithms, load-distribution is initiated by an under-loaded node (receiver), when it requests a task from an over-loaded node (sender).

Load sharing policies that use information about the average behavior of the system and ignore the current state, are called static policies. Static policies may be either deterministic or probabilistic. Policies that react to the system state are called adaptive or dynamic policies. Dynamic load sharing is an important system function designed to distribute workload among available processors and improve overall performance.

The principle advantage of using static policies is simplicity, since they do not require the maintenance and processing of system state information. Adaptive policies tend to be more complex, mainly because they do require information on the system's current state when making transfer decisions. However, the added complexity can significantly improve performance over that achievable with static policies.

Normally, distributed systems are heterogeneous; i.e. processors in the system operate at differing speeds. Therefore, research should address scheduling in heterogeneous environments. Schedulers for heterogeneous systems have special needs. For example, jobs encounter different execution times on different processors.

This paper studies the effects of load sharing on the job performance of distributed systems where half of the total number of the processors has double the speed of the others. Part of the jobs are dedicated to fast processors, while the remaining jobs are generic in the sense that they can be individually allocated to any processor.

Probabilistic, deterministic and adaptive policies are investigated. Performance estimates are obtained using simulation techniques.

In the probabilistic case, the scheduling policy is described by state independent branching probabilities. Dedicated jobs are dispatched randomly to fast processors with equal probability while the generic jobs are randomly dispatched to slow processors.

In the deterministic case, the routing decision is based on system state. Two different policies are examined for this case. In both policies, the dedicated jobs join the shortest of the fast processor queues. However, the first policy requires that generic jobs join the shortest queue of the slow processors while the second policy assigns generic jobs to the (slow or fast) processor expected to offer the least job response time. However, when a generic job is assigned to a fast processor, job start time depends on an aging factor.

In the adaptive case, variations of the three scheduling policies described above are employed. When fast processors become idle and generic jobs are waiting in slow processor queues, jobs migrate from heavily loaded slow processor queues to idle processors. This is a receiver initiated adaptive load sharing method. This balances the generic job load and can improve overall system performance.

Jobs transferred to remote processors incur communication costs. In this model, only queued jobs are transferred. We believe that the average transfer costs for non-executing jobs, although certainly non-negligible is quite low in comparison to average job processing costs. It is assumed that the job being executed is not eligible for transfer because doing so is too complex.

An obvious disadvantage of pre-emptive migration is the need to transfer memory associated with the migrated process; thus, migration costs for an active process is much greater than the cost of remote execution.

Several others have worked on load sharing and load balancing in heterogeneous systems. Cow and Kohler (1979) study load balancing in heterogeneous multiprocessor systems and introduce an approximate numerical method for analyzing models using deterministic routing policies. They show that a deterministic strategy that maximizes the expected throughput during the next interarrival interval gives the best performance.

Shenker and Weinrib (1989) study the optimal control of heterogeneous queuing systems. They propose heuristic policies for load sharing and routing. Simulation data on their policies suggest that their methods perform well over a wide range of system parameters.

Bonomi and Kumar (1990) consider a model consisting of several servers, each equipped with their own queue and operating at possibly different service speeds. Each server receives a dedicated arrival stream of jobs; there is also a stream of generic jobs that arrive at a job scheduler and which can be individually allocated to any of the servers. If the arrival streams are all Poisson and all jobs have the same exponentially distributed service requirements, the probabilistic splitting of the generic stream that minimizes the average job response time is such that it balances the server idle times in a weighted least-squares sense, where the weighting coefficients are related to the service speeds of the servers.

Mirchandaney et al. (1990) consider the impact of delay on the performance of heterogeneous distributed systems. They consider two types of heterogeneous systems: in type I systems external job arrival rates at nodes may differ and in type 2 systems the processing rates of the nodes may also differ. They use analytical models to estimate the performance. In order to facilitate analytical modeling, they assume that inter-arrival times and service times are exponentially distributed. They employ the non-preemptive first-come-first-served node scheduling policy. Their paper is directly relevant to the Dandamudi (1997) study. Dandamudi concentrates on non-exponential inter-arrival and service times, and uses preemptive round-robin node scheduling policy.

Maheswaran et al. (1999) study dynamic mapping heuristics for a class of independent tasks using heterogeneous computing systems. They consider on-line and batch mode heuristics. They introduce three different heuristics, one for batch and two for on-line. Their simulation results reveal that the choice of mapping heuristic depends on the structure of the heterogeneity among tasks and machines, the optimization requirements, and the arrival rate of the tasks.

Topcuoglu and al. (1999) propose two heuristics to schedule directed acyclic weighted task graphs on a bounded number of heterogeneous processors. They compare the performances of their algorithms against three previously proposed heuristics. The comparison study shows that their algorithms outperform previous approaches in terms of performance (schedule length ratio and speedup) and cost (time-complexity).

Heterogeneous distributed systems are also studied in Karatza (1994), and Karatza (2001). In the former paper no job migration is considered. The later paper does consider job migration, but it does not consider communication overhead. Also, in those two papers the system and workload models are different than the models that are examined here. They consider closed queuing network models with a fixed number of jobs. An open queueing network models the distributed processors in this paper, and migration overhead is taken into account.

The goal of this paper is to improve the performance of the generic jobs without seriously degrading the performance of the dedicated jobs. The performance of different load sharing policies is compared for various system loads. A comparative analysis of load sharing methods like this has not appeared in the research literature.

The paper is organized as follows. Section 2.1 specifies system and workload models, section 2.2 describes load sharing policies and section 2.3 presents the metrics employed in assessing the performance of the load sharing policies. Model implementation and input parameters are described in section 3.1 while the results of the simulation experiments are presented and analyzed in section 3.2. The final section offers conclusions and provides suggestions for further research.

2 MODEL AND METHODOLOGY

2.1 System and Workload Models

An open queuing network model of a distributed system is considered. $P = 16$ heterogeneous processors are available, each serving its own queue.

A high-speed network connects the distributed nodes. Configuration of the model is shown in Figure 1.

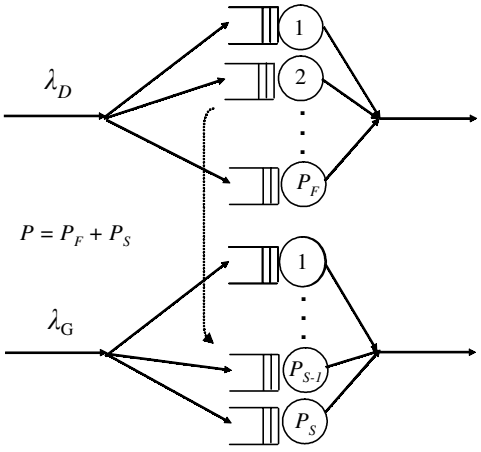

Figure 1: The Queuing Network Model

Half of the processors execute at double the speed of the others. We refer to the processors as either fast or slow. It holds that:

$$P_F = P_S = P/2,$$

where P_F (P_S) is the number of fast (slow) processors respectively.

There are two job classes. The jobs of one class are dedicated to fast processors, while jobs of the other class, referred to as generic, can be allocated to any processor. There is one arrival stream for the dedicated jobs and another arrival stream for the generic jobs.

Identical service requirements for the generic and dedicated jobs are assumed. Job classes contribute equally to the average service rate once a job has been assigned to a processor.

The jobs examined are highly independent. For example, once a job commences execution, no job ever idly waits for communication with (synchronizes with) other jobs.

Intuition and experience suggest that the best system and program performance can be achieved through a balanced use of the available resources, where both the overloading and under utilization of some resources is avoided. It is reasonable to assume that an optimal load balancing policy will try to equalize processor utilization. On one hand it seems as though dedicated jobs should not monopolize the fast processors. On the other hand, dedicated jobs should not be overtaken arbitrarily by the generic jobs.

The scheduling policies examined are probabilistic, deterministic and adaptive.

In the adaptive case, job migration from slow to fast processors is employed. This is a receiver-initiated adaptive load sharing method employed to improve the performance of generic jobs. The policy is initiated when a generic job is queued on a slow processor and a fast processor becomes idle. Only the migration of non-executing jobs is considered. Executing jobs are not eligible for transfer because of complexity issues.

An obvious disadvantage of preemptive migration is a requirement to transfer memory associated with the migrated process. Therefore, the migration cost for active processes can be greater than the cost of remote execution.

When a job is transferred from a slow to a fast processor for remote processing, the job incurs a communication cost. Only jobs that are waiting in the queues are transferred. The benefits of migration depend on migration costs.

The workload considered here is characterized by three parameters:

- The distribution of job arrival.
- The distribution of processor service time.
- The distribution of the migration overhead.

Dedicated job inter-arrival times and also generic job inter-arrival times are exponential random variables with a mean of $1/\lambda_D$ and $1/\lambda_G$ respectively (λ_D and λ_G are the arrival rates of dedicated and generic jobs respectively).

Processor service times are exponential random variables with mean $1/\mu_F = 1$ ($1/\mu_S = 2$) at the fast (slow) processors respectively.

The offered system load is the same for both classes. That is:

$$\lambda_D = 2 * \lambda_G,$$

because the fast processors are twice as fast as the slow processors.

We consider that the migration overhead is a uniformly distributed random variable in the interval [a, b]. The mean migration overhead is therefore (a+b)/2.

2.2 Job Scheduling Policies

We examine only non-preemptive job scheduling policies. We assume that the scheduler has perfect information when making decisions, i.e. it knows:

- The length of all processor queues.
- The queueing time of dedicated jobs in fast processor queues.

Next, the scheduling strategies we employ are described.

2.2.1 Probabilistic (Pr)

With this policy, a dedicated (generic) job is dispatched randomly to one of the fast (slow) processors with equal probability. The job dispatcher chooses one of the $P/2$ fast (slow) processors based on the outcome of an independent

trial in which the i^{th} outcome has probability $p_i = 1 / (P/2)$. Then the FCFS policy is applied.

2.2.2 Probabilistic with Migration of Generic Jobs (PrM)

In this case, jobs are assigned to processor queues in the same way as in the *Pr* case. However, when a fast processor becomes idle and generic jobs are waiting in the slow processor queues, a job migrates from the most heavily loaded slow processor to the idle fast processor. This is a receiver-initiated algorithm, as load-distributing activity is initiated from an under-loaded node (receiver), which tries to get a job from an over-loaded node (sender).

2.2.3 Shortest Queue (SQ)

With this strategy, a dedicated job is assigned to the shortest queue of fast processors, while a generic job is assigned to the shortest queue of the slow processors. Each job is inserted into its assigned queue in the order of its arrival.

2.2.4 Shortest Queue with Migration of Generic Jobs (SQM)

This is a variation of SQ, where migration takes place in the same manner as in PrM.

2.2.5 Least Expected Response Time for Generic Jobs-Maximum Wait for Dedicated Jobs (LERT-MW)

With this method, dedicated jobs join the shortest queue of the fast processors. The generic jobs join the queue of the processor (slow or fast) that offers the least expected response time.

This policy is described formally as follows.

Assume the system is in the state $s = (l_1, l_2, ..., l_P)$, where l_i, $(i = 1, 2, ..., P)$ are the number of jobs in the i_{th} queue including the job in service.

An arriving generic job is sent to queue i, $(i \in \{1, 2, ..., P\})$ if the following relation holds:

$$(l_i + 1) * (1/\mu_i) = \min ((l_k + 1) * (1/\mu_k)) \text{ for } k = 1, 2, ..., P,$$

where μ_i, $(i = 1, 2, ..., P)$ is the mean service time of processor i.

If the minimum value is not unique, the job dispatcher selects the processor with the smallest mean service time. The minimum job response time policy is based on a user's view of how to improve performance, since $(l_k+1) * 1/\mu_k$ is the expected response time of the next job on processor k. In heterogeneous systems, the queuing of jobs is often preferable when the only available processor is relatively slow.

There is a trade off between extra time spent in a queue waiting for a fast processor, and extra time spent being serviced on a slow processor.

With this scheme dedicated jobs share the fast processors with generic jobs. However, dedicated job sequencing should be preserved as much as possible to achieve execution fairness. For this reason, if a dedicated job has been waiting for more than a configurable period of time *MW*, it is scheduled before any generic job ahead of it in the queue. Otherwise, FCFS is applied to all jobs in the queue.

2.2.6 Least Expected Response Time for Generic Jobs-Maximum Wait for Dedicated Jobs with Migration (LERT-MWM)

This is the migratory version of LERT-MW.

2.2.7 General Remarks

All six of the above scheduling schemes have merit.

Pr is the simplest method since the scheduler creates only a small amount of overhead when generating random numbers.

The SQ method requires knowledge regarding half of the queues on job arrival (sub-global information).

The LRT-MW policy needs global (sub-global) information on queue lengths for the generic (dedicated jobs) respectively, and it also requires additional information about the time dedicated jobs have been waiting in a queue.

When using migratory versions of these policies, the scheduler requires additional load information to decide when a fast processor becomes idle after a job departure. The collection of global (and sub-global) information requires a non-trivial amount of overhead, but it is necessary to implement even a moderately effective scheduler. It is obvious that the Pr policy is easier to implement because it requires less overhead.

When a transferred job arrives at the destination node, the node must accept and process the transferred job even if the state of the node at that instance has changed since probing.

2.3 Performance Metrics

We consider the response time of a random job as the time interval from the arrival of a job in a processor queue to service completion (at the same processor or at a different processor in case of migration).

Notations used in this paper appear in Table 1.

Table 1: Notations

P	Number of processors
P_F (P_S)	Number of fast (slow) processors
λ_D (λ_G)	Mean arrival rate of dedicated (generic) jobs
$1/\mu_F$ ($1/\mu_S$)	Mean fast (slow) processor service time
RT_D (RT_G)	Mean Response Time of dedicated (generic) jobs
RT	Mean Response Time of all jobs
[a, b]	Interval within which migration overhead is uniformly distributed
MW	Threshold for maximum wait in the LERT-MW case
DRT_D (DRT_G)	Relative increase in RT_D (relative decrease in RT_G) when a migratory version of a policy is employed

3 SIMULATION RESULTS AND DISCUSSION

3.1 Model Implementation and Input Parameters

The queuing network model described above is implemented with discrete event simulation (Law and Kelton 1991) using the independent replication method. For every mean value, a 95% confidence interval is evaluated. All confidence intervals are less than 5% of the mean values.

Because $1/\mu_F = 1$ and $1/\mu_S = 2$, we examine the following cases for mean inter-arrival time of dedicated and generic jobs:

$$1/\lambda_D = 0.13, \text{ and } 1/\lambda_G = 0.26,$$
$$1/\lambda_D = 0.14, \text{ and } 1/\lambda_G = 0.28,$$
$$1/\lambda_D = 0.15, \text{ and } 1/\lambda_G = 0.30,$$

which correspond to the following arrival rate cases:

$$\lambda_D = 7.69, \text{ and } \lambda_G = 3.85,$$
$$\lambda_D = 7.14, \text{ and } \lambda_G = 3.57,$$
$$\lambda_D = 6.67, \text{ and } \lambda_G = 3.33.$$

When the dedicated jobs are evenly distributed among the fast processors, the expected mean fast processor utilization is $(\lambda_D / P_F) / \mu_F$. This results in mean fast processor utilization of:

$$0.96, 0.89, \text{ and } 0.83,$$

respectively for the above three cases of λ_D.

It is also obvious that the mean slow processor utilization is the same as that of the fast processors, when generic jobs are evenly distributed among slow processors. However, migration of generic jobs results in a decrease of slow processor utilization and an increase in fast processor utilization.

For the interval [a, b], we consider $a = 0.09$ and $b = 0.11$ (i.e., average migration overhead is 10% of the mean processor service time).

$MW = 2$ is considered for achieving fairness in the execution of dedicated jobs. This is a reasonable assumption since mean service time at the fast processors is 1.

3.2 Performance Analysis

The following results summarize overall model performance (Figures 2-13).

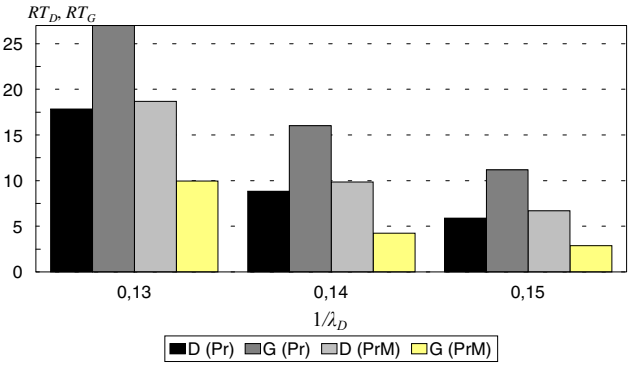

Figure 2: RT_D , RT_G versus $1/\lambda_D$ (PR and PRM cases)

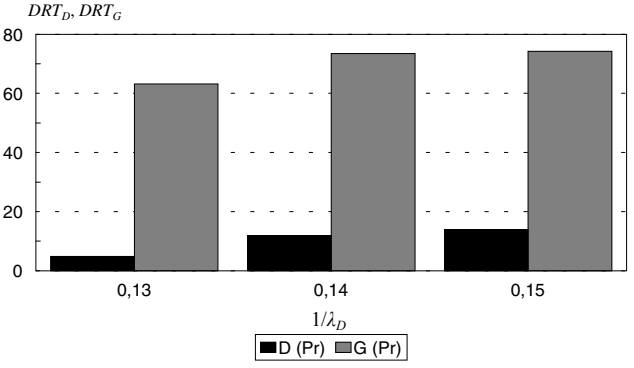

Figure 3: DRT_D , DRT_G versus $1/\lambda_D$ (PR and PRM cases)

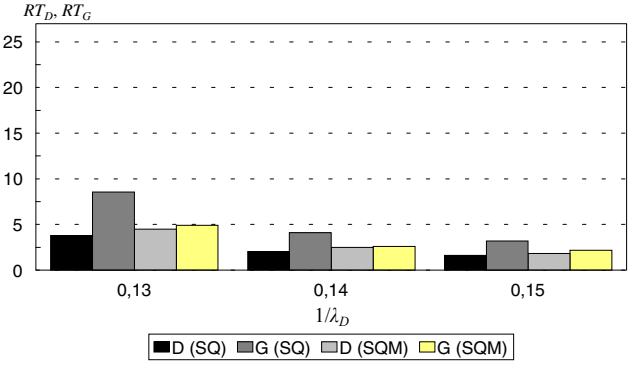

Figure 4: RT_D, RT_G versus $1/\lambda_D$ (SQ and SQM cases)

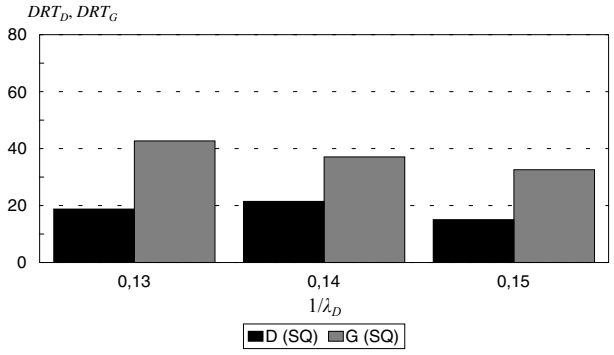

Figure 5: DRT_D, DRT_G versus $1/\lambda_D$ (SQ and SQM cases)

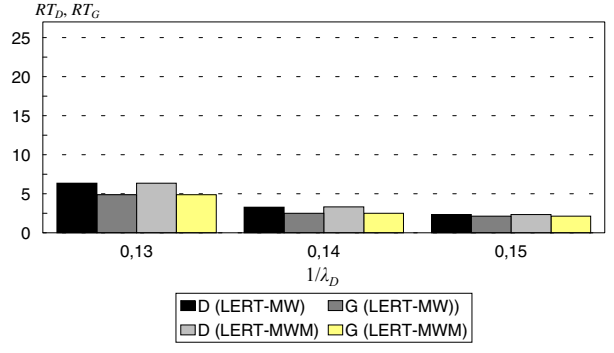

Figure 6: RT_D, RT_G versus $1/\lambda_D$ (LERT-MW and LERT-MWM cases)

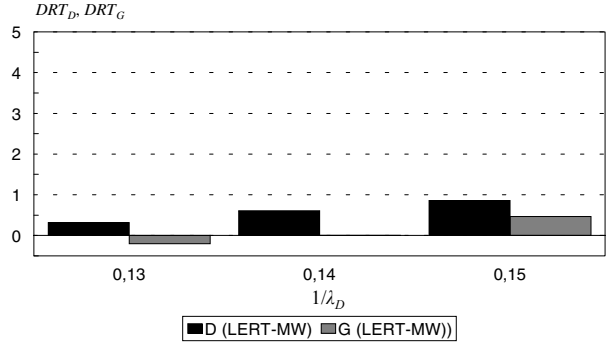

Figure 7: DRT_D, DRT_G versus $1/\lambda_D$ (LERT-MW and LERT-MWM cases)

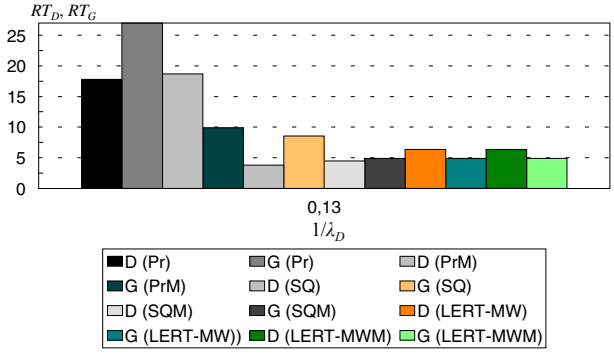

Figure 8: RT_D, RT_G, $1/\lambda_D = 0.13$

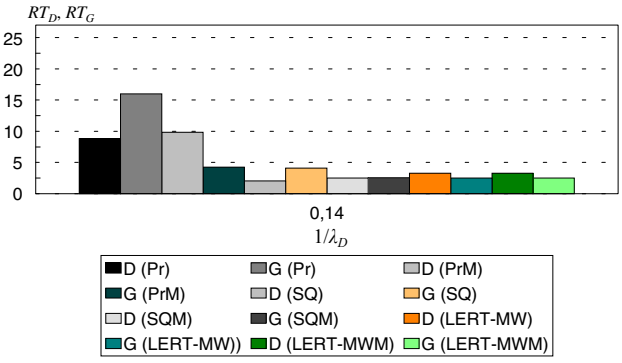

Figure 9: RT_D, RT_G, $1/\lambda_D = 0.14$

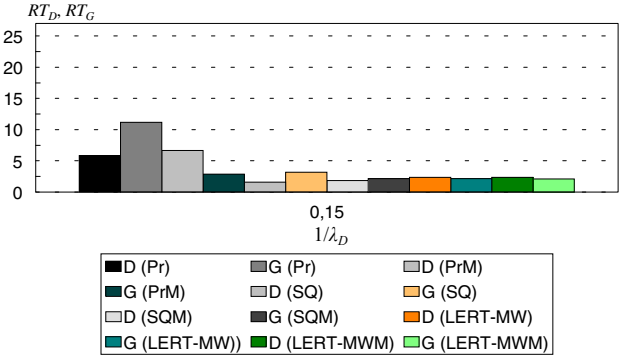

Figure 10: RT_D, RT_G, $1/\lambda_D = 0.15$

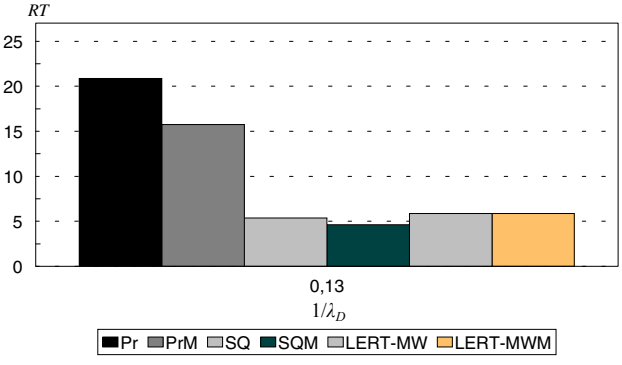

Figure 11: RT, $1/\lambda_D = 0.13$

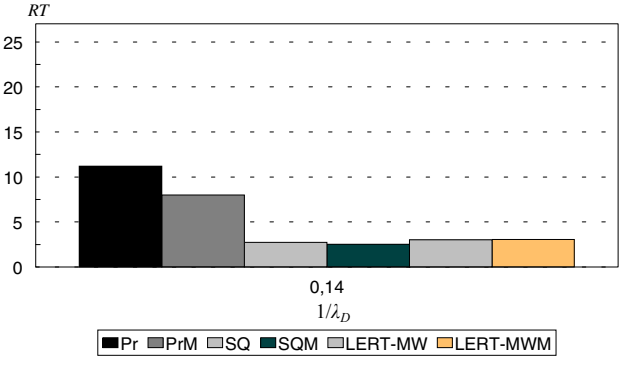

Figure 12: RT, $1/\lambda_D = 0.14$

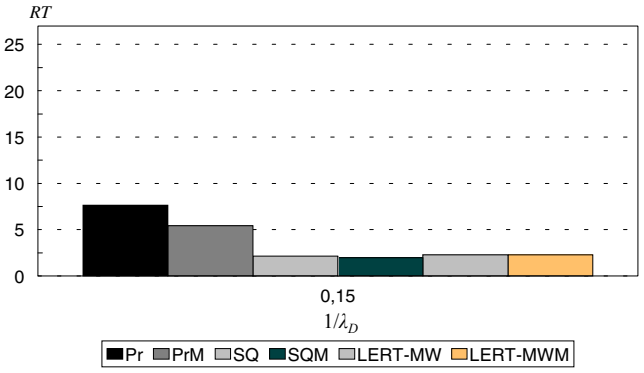

Figure 13: RT, $1/\lambda_D = 0.15$

Figures 2, 4 and 6 show that for all loads examined, the migratory version of each strategy improves the performance of the generic jobs. This is due to the fact that when generic jobs migrate, they are served by fast processors. Therefore, they have shorter response times than they would if they were served by slow processors. On the other hand, when a dedicated job arrives at a fast processor, which is already serving a generic job, the dedicated job incurs queuing delays. The values of RT_D and RT_G for all routing strategies that we examine are presented in Figures 8, 9, and 10 respectively for each of the $1/\lambda_D$ cases.

The relative increase in RT_D and the relative decrease in RT_G due to using a migratory version of a method is shown in Figures 3, 5, and 7. These Figures show that the performance improvement for generic jobs due to job migration is much higher than the performance degradation for dedicated jobs in the probabilistic case. In all cases that we examine, the probabilistic case presents the largest DRT_G. This is because the Pr strategy results in unbalanced processor queues and therefore introduces migration opportunities for the generic jobs. DRT_G increases as system load decreases. This is because it is more probable for the processors to be idle at low loads than at high loads and so there are more possibilities for generic job migration at low load. For a mean inter-arrival time of 0.15, the relative decrease in response time for generic jobs is 74% while for mean inter-arrival time 0.13, D_{RG} is 63%. DRT_D is 5%, 12%, and 14% in the $1/\lambda_D = 0.13$, 0.14, and 0.15 cases respectively. The reason dedicated jobs are influenced to a lesser degree than the generic jobs in the PRM case is because the generic jobs are only transferred to idle fast processors. DRT_D is higher at low loads because there are more opportunities for generic job migrations.

In Figures 3 and 5 it appears that the dedicated job performance degradation due to generic job migration for all system loads is more serious in the SQ case than in the Pr. These Figures also indicate that generic jobs benefit from migration in the SQ case to a smaller degree than in the Pr case. This is because fast processor queues are balanced in the SQ case. Therefore, there are less opportunities for the generic jobs to find idle fast processors in the SQM case than in PrM. On the other hand, dedicated job response time in the SQ case is much smaller than in Pr. A generic job that migrates to an idle fast processor prevents a subsequent dedicated job from using this idle processor. This is the reason that a larger DRT_D appears in the SQM case than in PrM. However, Figures 8, 9, and 10 show that for all loads of the six methods that we examine, the Pr and PrM methods yield the largest mean response time for both dedicated and generic jobs.

Figures 6 and 7 show for all loads dedicated jobs perform worse than generic jobs in the LERT-MW and LERT-MWM cases. This is because generic jobs choose the processor which offers them the shortest expected response time from among all system processors, while dedicated jobs only choose one of the fast processors. The migration of generic jobs only marginally effects the performance of dedicated and generic jobs because the shortest expected response time criterion results in a smaller number of opportunities for job migration than the other routing methods do.

Figures 11-13 reveal that the overall performance in terms of mean response time of all jobs (dedicated and generic) is better with the SQ and SQM methods. Regarding these two policies SQM yields the smallest mean response time. The relative difference in performance between these two methods is 14%, 8%, and 9% in the $1/\lambda_D = 0.13$, 0.14, and 0.15 cases respectively.

In terms of overall performance, the worst methods are the two probabilistic policies, Pr and PrM. The migration of jobs in the PrM case improves significantly the overall performance as compared to Pr. This is because there are opportunities for job migration due to unbalanced processor queues with the Pr policy. Therefore, the generic job migration case results in effective usage of idle system resources. For $1/\lambda_D = 0.13$, 0.14, and 0.15 the relative difference in performance between PrM and Pr methods is 24.5%, 29% and 29% respectively. However, the overall performance of the PrM strategy differs significantly from the performance of SQ, and SQM. The relative difference in performance between PrM and SQ (SQM) is 66%, 65%, 60% (71%, 68%, 64%) in the $1/\lambda_D = 0.13$, 0.14, 0.15 cases respectively.

The LERT-MW and LERT-MWM strategies perform almost the same. The relative difference in performance between LERT-MWM and SQM is 21%, 17%, 15% respectively for the $1/\lambda_D = 0.13$, 0.14, 0.15 cases.

4 CONCLUSIONS AND FURTHER RESEARCH

This paper studies load sharing in heterogeneous distributed systems. The objective is to obtain good overall system performance while maintaining the fairness of individual job classes. Simulation is used to generate comparative results.

The performance of six load sharing policies is studied (Pr, PrM, SQ, SQM, LERT-MW, and LERT-MWM) for

various mean job inter-arrival times. Migration overhead is taken into account. Simulation indicates the following:

- As far as overall performance is concerned, the SQ and SQM methods perform better than all other methods. SQM performs better than SQ. However, the superiority of SQM over SQ depends on system load.
- The superiority of SQ and SQM over the other methods also depends on system load and is more significant at high loads.
- SQM is the best method when individual job class performance and fairness is important.

A priori knowledge of job execution time is not considered. Future research should include the use of a priori information when making load sharing decisions so that jobs with very small service times are not migrated.

REFERENCES

Bonomi, F., and A. Kumar. 1990. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Transactions on Computers* 39 (10): 1232-1250.

Cow, Y.-C., and H. W. Kohler. 1979. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Transactions on Computers* 28 (5): 354-361.

Dandamudi, S. 1997. The effect of scheduling discipline on dynamic load sharing in heterogeneous distributed systems. In *Proceedings of the 5th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, 17-24. Los Alamitos, California: Institute of Electrical and Electronics Engineers Computer Society.

Karatza, H. D. 1994. Simulation study of load balancing in a heterogeneous distributed system model. *International Journal of Modelling and Simulation* 14 (1): 28-33.

Karatza, H. D. 2001. Job scheduling in heterogeneous distributed systems. *Journal of Systems and Software* 56 (3): 203-212.

Law, A., and D. Kelton. 1991. *Simulation modelling and analysis*. New York: McGraw-Hill.

Maheswaran, M., S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. 1999. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the Eighth Heterogeneous Computing Workshop*, 30-44. Los Alamitos, California: Institute of Electrical and Electronics Engineers Computer Society.

Mirchandaney, R., D. Towsley, and J. Stankovic. 1990. Adaptive load sharing in heterogeneous systems. *Journal of Parallel and Distributed Computing* 9 (4): 331-346.

Shenker, S., and A. Weinrib. 1989. The optimal control of heterogeneous queueing systems: a paradigm for load-sharing and routing. *IEEE Transactions on Computers* 38 (12): 1724-1735.

Topcuoglu, H., S. Hariri, and M-Y. Wu. 1999. Task scheduling algorithms for heterogeneous processors". In *Proceedings of the Eighth Heterogeneous Computing Workshop*, 3-14. Los Alamitos, California: Institute of Electrical and Electronics Engineers Computer Society.

AUTHOR BIOGRAPHIES

HELEN D. KARATZA is an Associate Professor in the Department of Informatics at the Aristotle University of Thessaloniki, Greece. Her research interests include Computer Systems Performance Evaluation, Multiprocessor Scheduling, Mobile Agents and Simulation. Her email and web addresses are <karatza@csd.auth.gr> and <http://agent.csd.auth.gr/~karatza>.

RALPH C. HILZER is a Professor of Computer Science at the California State University, Chico, USA. His research interests are in the areas of Operating Systems, Parallel Processing, Languages and Compilers. His email and web addresses are <rhilzer@csuchico.edu> and <http://www.ecst.csuchico.edu/~hilzer>.