Dynamic versus Adaptive Processor Allocation Policies for Message Passing Parallel Computers: An Empirical Comparison *

Jitendra Padhye[1] and Lawrence Dowdy[2]

[1] Department of Computer Science, University of Massachusetts at Amherst, ***
Amherst, MA 01003.
[2] Department of Computer Science, Vanderbilt University,
Nashville, TN 37235.

Abstract. When a job arrives at a space-sharing multiprocessor system, a decision has to be made regarding the number and the specific identities of the processors to be allocated to it. An adaptive policy may consider the state of the system at arrival time but it does not allow preemption of any of the running jobs. A dynamic partitioning policy may preempt one or more of the currently running jobs to accommodate the new arrival. In this paper performance of dynamic and adaptive policies is investigated experimentally on a message passing architecture (Intel Paragon). The workload model is based on matrix computation applications commonly found on large systems used for scientific programming. Results are reported for single and multiclass cases. A sensitivity analysis with respect to workload speedup characteristics is presented. Our results show that if the preemption overheads are kept low, dynamic polices result in noticeable improvement in overall performance of the system.

1 Introduction

Multiprogramming is a common way of improving performance of large multiprocessor systems. Most common parallel applications have sublinear speedup curves and hence can not take full advantage of all system processors. For a heavily used system it is not uncommon to have several parallel jobs waiting to use the multiprocessor. In such cases, performance can be improved by sharing the multiprocessor among all or some of the waiting jobs. This can be achieved via either space sharing [SRSDS94] or time sharing [GTU91].

Under space sharing scheduling polices, an incoming job is assigned to a subset of the total available processors. Thus, multiple jobs can be active within the multiprocessor at the same time. Space sharing policies can either be dynamic or adaptive. Several issues such as application speedup characteristics, underlying multiprocessor architecture, and special requirements of certain application

* Supported in part under sub-contract 19X-SL131V administered through the Mathematical Sciences Section of Oak Ridge National Laboratory.
*** This work was done at Vanderbilt University.

workloads are among the factors affecting performance of processor allocation strategies for multiprogramming parallel systems. In this paper, effects of some of these factors on the performance of dynamic and adaptive space sharing policies are investigated empirically.

Several dynamic and adaptive processor allocation policies have been discussed in the literature [MEB88, TG89, PD89, LV90, DCDP90, ZM90, GTU91, ZB91, MVZ93, SST93, RSDSC94, SEV94, CMV94, MZ94]. Performance analysis of dynamic space sharing scheduling policies has been largely based on simulation studies and Markovian analysis of small systems. For a simulation study to be accurate and realistic, detailed knowledge of various parameters of the system under consideration is necessary. Also, validation of the simulation models is difficult. Detailed Markovian analysis of complex scheduling policies to verify simulation models is possible only for small systems (e.g. less than 10 processors) [SRSDS94, DCDP90, MEB88]. For larger systems, simplifying assumptions have to be made, resulting in a loss of accuracy.

Experimental studies on dynamic processor partitioning policies have been done mainly for shared memory architectures [GTU91, TG89]. In this paper, experimental analysis of dynamic processor partitioning policies for a message passing architecture is presented. To this end, two dynamic processor partitioning policies based on those discussed in [MZ94] and one adaptive policy presented in [RSDSC94] are implemented on the Intel Paragon. Two workload programs are used to compare behavior of the implemented scheduling policies. One is a synthetic workload program designed to emulate various speedup curves and other characteristics of common scientific applications. The flexibility of the synthetic workload allows easy emulation of a wide variety of load conditions for comparing the scheduling policies. The other workload program is based on a parallel, preemptible, distributed memory version of a matrix conjugate gradient program. This program is representative of typical scientific workloads and has been used for similar purposes in the past [BMSD95]. Our results show that if the preemption overheads are kept low, dynamic polices result in noticeable improvement in the performance of the system. This result is not surprising theoretically, however, experimental validation studies have been lacking. This study is an effort to fill that gap.

The rest of the paper is organized as follows. Section 2 describes the dynamic and adaptive scheduling policies and the two workload programs used for this study. A brief description of the Intel Paragon architecture is also presented. The results of the study are presented in Section 3. Section 4 presents the conclusions and directions for future work.

2 Policies and Workloads

This section briefly describes the three scheduling policies studied in this paper, the two workload programs used for comparisons, and the architecture of the Intel Paragon. Due to space constraints, this section provides only a brief overview of the actual implementation. More details may be found in [P96].

2.1 The Intel Paragon

The Intel Paragon supercomputer consists of several nodes connected in a mesh configuration. The computer used for this study has 66 nodes connected in a 11x6 matrix. Several nodes are dedicated to special tasks such as disk and network control. Each node consists of two Intel i860 processors sharing 16 MB of memory. One of the processors runs the application program while the other acts as a communication co-processor. To route messages between the nodes, a wormhole routing protocol is used. The operating system conforms to OSF/1 standards. Although the machine is capable of supporting MIMD (or MPMD) computing model, most applications use the SPMD model for computing. The workloads for this study use the SPMD model. A detailed description of the Paragon architecture may be found in [INT93].

2.2 The Adaptive Scheduling Policy

The adaptive scheduling policy chosen for this study is the Robust Adaptive (RA) scheduling policy described in [RSDSC94]. This policy has been shown to have better performance than several other non-preemptive scheduling policies. The RA scheme actually represents a suite of scheduling policies rather than a single scheduling algorithm. The version chosen here is a representative one and has been used in subsequent studies [SRSDS94]. The main feature of the RA policy is that it adapts to load changes over a period of time. Two identical jobs may be allocated different sized partitions, depending upon the system state when each job arrives. When a new job arrives in the system, it is placed at the end of a queue of jobs awaiting service. The scheduler calculates a "target" partition size using the following formula:

$$max\left(1,\ \left\lfloor \frac{\text{Total number of processors in the system}}{\text{Number of jobs waiting in the queue}} \right\rfloor\right)$$

The scheduler does not schedule any jobs until at least "target" number of processors are free. It then starts allocating "target" number of processors to each job in the queue. If the processors can be allocated, then the job is started immediately. The target is not recomputed during this time. When no more jobs can be scheduled, either due to an empty queue or due to a lack of sufficient free processors, the scheduler stops scheduling new jobs and recomputes a new target. The target recalculation allows RA to adapt to changes in the load condition. A higher load results in a longer queue, which results in a smaller target which in turn means that more jobs are scheduled.

2.3 Dynamic Partitioning Policies

Dynamic partitioning policies can interrupt currently executing jobs in order to redistribute processors among the jobs because of job arrivals and/or departures. Thus, dynamic scheduling policies can quickly adapt to transient changes in the workload flow. The two dynamic processor partitioning policies considered in this study are based on policies discussed in [MZ94]. Implementing preemption on a message passing architecture is a non-trivial task. The steps taken to ensure "safe" preemption of jobs while keeping the preemption overhead low, are briefly described in Section 2.4.

Equipartitioning. This policy tries to equally allocate the available processors among all jobs that are present in the system. On each new arrival or departure, all the currently executing jobs are preempted. The new partition size for each job is calculated using the following formula:

$$max\left(1, \left\lfloor \frac{\text{Total number of processors in the system}}{\text{Number of ready jobs}} \right\rfloor\right)$$

The number of ready jobs includes the new arrivals and the jobs that have been preempted. Since at least one processor has to be allocated to each job, it may not be possible to restart all the ready jobs. However, the scheduler always restarts all the preempted jobs.

The policy suffers from two major disadvantages. First, a job can be preempted several times [4] under this policy. Even when individual preemptions are not very costly, the total cost can be prohibitive. Secondly, this policy suffers from *synchronization delays.* Since some of the currently running jobs may finish during the process of repartitioning, [5] the scheduler has to wait until all the processors in the partition are released before calculating the new partition size for each job. A job that has been preempted is forced to wait until all the remaining running jobs free their partitions. This synchronization delay may be a source of significant overhead.

Folding. This policy preempts at most one of the currently running jobs as a result of an arrival or a departure. Hence, the policy does not suffer from excessive synchronization delays. At each new arrival, the scheduler checks to see if there are free processors. If there are free processors in the system, the new job is started on all of them. No preemptions take place in this case. If there are no free processors then the scheduler checks the size of largest partition currently allocated to a job. If this size is greater than 1, then the job running on the largest partition is preempted and half of the processors are given to the incoming job. This is called *folding.* If there are multiple candidates for folding then the implementation can select any one of them. The implementation used for this study preempts the most recently arrived job in the event of such a tie. At any time, if the system has any idle processors, and no jobs are waiting in the ready queue, then the job executing on the smallest partition is preempted and the idle processors are merged with the job's old partition and the job is restarted on the new partition thus formed. This is called *unfolding.* If there

[4] Theoretically, once per every subsequent job arrival and job departure.

[5] These are the jobs that finish *after* the scheduler has sent out preemption requests. The scheduler will not restart these jobs.

are several candidates for unfolding, the implementation may choose any one of them. The implementation used for this study unfolds the most recently arrived job in the system.

2.4 Minimizing the Cost of Preemption

A parallel job may not be preempted "safely" [6] at any arbitrary time during its execution. This may happen due to several reasons (e.g. the job might be executing a piece of non-reentrant code). At other times, it might be safe to preempt the job, but the cost of preemption might be high. This may happen when the job is in the middle of a distributed computation and preemption requires realignment of data. It is difficult for the scheduler to determine when it is easy (i.e. safe as well as cheap) to preempt a job. Hence, this implementation requires that all the jobs have certain *break points*, where it is safe to preempt the job and the preemption overheads can be kept low. A job responds to a preemption request by the scheduler only when it is at a breakpoint.

The processor allocation routines of all the dynamic schedulers implemented for this study observe the *subset* restriction to minimize any data redistribution overhead that the job might incur as a result of preemption. The *subset* restriction is defined as follows. If P_1 and P_2 represent the sets of processors in two successive partitions allotted to a job then:

$$\begin{aligned} P_1 \cap P_2 &= P_1 \text{ if } |P_1| \leq |P_2| \\ &= P_2 \quad \text{otherwise} \end{aligned}$$

This restriction helps to minimize the data transfers required during redistribution. In addition, the routine tries to allocate contiguous partitions. The preemption process is carried out in four steps.

1. The scheduler sends preempt requests to all the jobs it wishes to preempt. Whenever each job reaches its next breakpoint, it informs the scheduler that it is ready to be preempted.
2. Once the scheduler receives such replies from all the jobs to whom such requests are sent[7], it calculates the new processor allocations. All the jobs that will be preempted are informed of their new partitions.
3. Depending on the size of the new partition, the job executes one of the following two steps.
   (a) If the size of the new partition is smaller than the size of the previous partition, then the job is termed as *LOSER*. A *LOSER* job determines the subset of its old partition that forms the new partition. The data is redistributed so that all the application data is stored on the processors in this subset.
   (b) If the size of the new partition is larger than the old one, then the job is termed a *WINNER*. A *WINNER* job has to wait until the extra processors are allocated to it to do its data redistribution.

[6] In this context, *safety* essentially implies ensuring *correctness*.

[7] Or, if the job terminates.

4. Once the data redistribution is finished, the jobs release their processors and the scheduler restarts the jobs on their new partition after it finishes its internal bookkeeping operations. After the restart, the *WINNER* jobs redistribute their data and start execution. The *LOSER* jobs, on the other hand, having already finished their data redistribution, resume execution immediately.

The entire preemption sequence is implemented as a series of calls to a set of library routines. These routines are independent of the nature of the application program and the scheduling policy.

2.5 Workload Description

This section describes the two workloads used for the experimental study. The first is a synthetic workload, designed to have enough flexibility to emulate various speedup curves. Number of computations, amount of communication and preemption overheads are all easily adjustable. The second workload is based on the Conjugate Gradient method for matrices. The program is a representative example of the scientific workload and has been used for similar purposes in other studies [BMSD95]. [8]

The Synthetic Workload. The synthetic workload is designed to be flexible enough to generate various speedup curves by varying workload parameters. The workload is capable of spawning as many tasks as the number of allocated processors. When the workload is started on a given partition, the lowest numbered node in the partition is selected as the co-ordinator. The workload operates in phases. Each phase consists of three subphases, namely, the *broadcast communication subphase*, the *compute subphase*, and the *collect communication subphase*. In the broadcast communication subphase, the co-ordinator broadcasts a message to each node in the partition. This message contains initialization parameters for that phase and other problem related data. In the compute subphase, each node in the partition does a certain amount of computation. There is no communication among the nodes during this subphase. During the collect communication subphase, each node in the partition sends back a message to the co-ordinator node. This message contains partial results and other synchronization data. At the end of each phase the nodes undergo a barrier synchronization. After the barrier synchronization, the job is in a safe state for preemption. At this point the co-ordinator checks to see if a preemption request has arrived from the scheduler. If such a request is present, the preemption sequence is executed and the job preempts. Otherwise, the next phase of the communicate-compute-communicate cycle is started. The amount of computation and communication during each subphase can be specified individually for each phase. The amount

[8] Other models of large, parallel, scientific workloads and arrival processes exist, and it will be interesting to study the effectiveness of dynamic partitioning policies for various workload models.

of data redistribution to be done in the event of preemption can also be specified. Changing the amount of computation or communication done in each phase yields workloads with different speedup curves. The preemption cost can be varied by changing the data redistribution required in the event of preemption or by changing the number or the length of the phases. Note that the cost of a preemption is reflected in the waiting time of the jobs that are waiting in the arrival queue and synchronization delay for the jobs that have already been preempted. The speedup curves generated by the synthetic workload having 50 phases appear in Figure 1. For each curve, the number of *total* computations done per phase is indicated in the legend. For all curves the message size during broadcast as well as in the collect communication phase is 32 bytes.

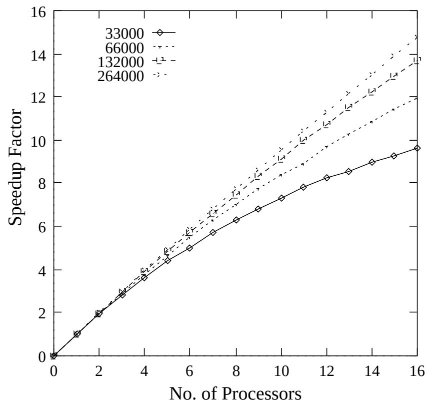

Fig. 1. Speedup Curves for the Synthetic Workload

Conjugate Gradient Workload. The program implements the conjugate gradient method for matrices. The task graph of the program is similar to many scientific and engineering applications found on large multiprocessor systems. More details about the program can be found in [BMSD95]. The program executes a specified number of iterations or until a user specified accuracy is achieved. For test purposes, the accuracy testing feature is disabled and the program continues for a specified number of iterations. During each iteration several distributed computations are carried out, requiring significant communication among processors belonging to the job. At the end of each iteration, all the processors undergo barrier synchronization. After the synchronization, the job is in a safe state to be preempted and the co-ordinator (lowest numbered processor) checks for preemption request. Data redistribution involves redistributing data contained in five different arrays or matrices. Thus, the preemption overhead involved can be high. The speedup curves for the CG workload are shown in the Figure 2. The

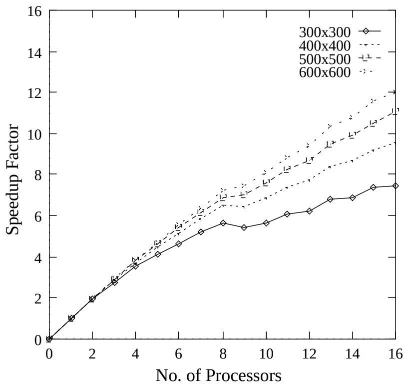

Fig. 2. Speedup Curves for the Conjugate Gradient Workload

different speedup curves are generated by using different matrix sizes. For each curve, the legend indicates the number of rows and columns of the matrix used as the data set for the program. Each CG program performs 250 iterations on the data matrix.

3 Results

This section is organized as follows. In Section 3.1, the performance of the Equipartitioning policy is compared against that of the Folding policy. In Section 3.2, sensitivity analysis results of the relative performance of RA and Folding policies are presented. Factors considered include constant versus exponential demand, speedup curves, and multiclass workloads. In Section 3.3, the performance of the RA and the Folding policies is compared on a larger multiprocessor to show that the results are scalable. For all policies, the smallest possible partition size is one processor. For the dynamic policies, the number of times a job may be preempted is bounded only by the number of "safe" preemption points in the job. Experimental results indicate that the performance of the restricted version of the dynamic scheduling policies (where the number of preemptions of a job and/or the minimum partition size are bounded by anything other than the base limits described above) depends strongly on the speedup characteristics of the workload. Hence only the comparisons between the base versions of the policies are presented.

3.1 Equipartitioning versus Folding

Figure 3 compares the performance of the Equipartitioning and the Folding scheduling policies. The workload used is the CG program with a 500x500 matrix. Each preemption requires redistribution of approximately 2 megabytes of data. The speedup curve of the program is shown in Figure 2. The interarrival times are exponentially distributed, while the workload demand is constant, (i.e. each program in the workload stream is identical to all others). It may be seen that the Folding policy exhibits a better response time than the Equipartitioning policy at all arrival rates, and the difference between them increases with increasing arrival rates. Folding allocates more processors to each job than Equipartitioning at all arrival rates. Thus, jobs have lower execution times under Folding. In addition, the average number of preemptions per job is higher under Equipartitioning than under Folding. Also, Equipartitioning suffers from higher synchronization delays. These factors result in increased waiting time (and hence increased response times) for jobs under Equipartitioning. Folding continues to significantly outperform Equipartitioning for various speedup curves and workload mixes. Hence, the remaining results in this section compare only RA versus Folding.

3.2 RA versus Folding

Constant versus Exponential Workload Demands. The synthetic workload is used to compare the performance of the policies under constant and exponential workload demands, instead of the CG program, since it is easier to vary the demand imposed by the synthetic workload on the multiprocessor by varying the number of computations done in each phase. Figure 4 compares the performance of the RA and the Folding scheduling policies using the synthetic workload. The workload performs 66000 floating point multiplications per phase. When preempted, 0.5 megabytes of data require redistribution. The speedup curve of the workload is shown in Figure 1. The interarrival times are exponentially distributed, while the workload demand is constant.

It can be seen that the Folding gives a better response time than RA at all arrival rates. The maximum difference in the response times is approximately 17%, achieved at higher arrival rates. The better performance of the Folding policy can be attributed to the reduction in the waiting time of the jobs under Folding. The Folding policy schedules a newly arrived job as soon as it can preempt one of the running jobs. If there are idle processors, then the arriving job does not have to wait at all. RA makes a newly arrived job wait until it can allocate a partition of current target size [9]. Except at very low arrival rates, the target partition size usually can not be allocated until at least one of the currently executing jobs finishes execution. Thus the jobs experience higher waiting time under RA. Folding is able to allocate more processors to each job than RA at higher arrival rates. However, job execution times are higher under Folding than under RA. This is due to preemption overheads the jobs experience under Folding. This can be seen from two graphs in Figure 4: 1) when execution time is plotted as a function of average number of processors per job and 2) when execution time is plotted as a function of the arrival rate. The maximum increase in the execution time is approximately 17%. It should also be noted that

[9] In other words, Folding is work-conservative, while RA is not.

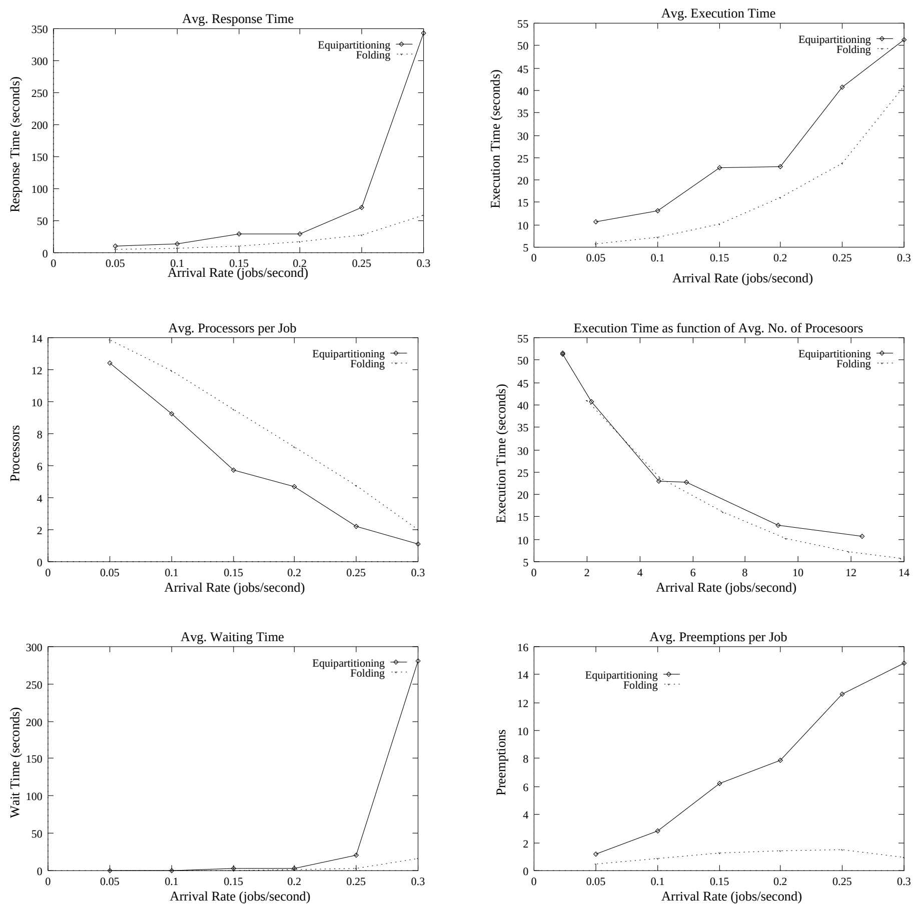

Fig. 3. Equipartitioning versus Folding

these preemption overheads are the result of a relatively small number of preemptions. On average, fewer than 1.5 preemptions per job occur at any arrival rate. It is important to note that the preemption overheads also depends upon the sizes of partitions before and after the preemptions. These sizes determine *how many* processors will participate in the preemption and, hence, affects the data redistribution overhead.

Figure 5 compares the performance of the RA and the Folding scheduling policies when the workload demand is exponentially distributed. The synthetic workload program is used for this comparison as well. All the workload parameters are the same as in the previous comparisons, except that the workload demand is exponentially distributed. It is noted that the improvement in the response time (maximum 25%) is better than the improvement (maximum 17%) achieved when the workload demand was constant. Due to its dynamic nature, Folding is able to handle the variations in the workload demand better than RA.

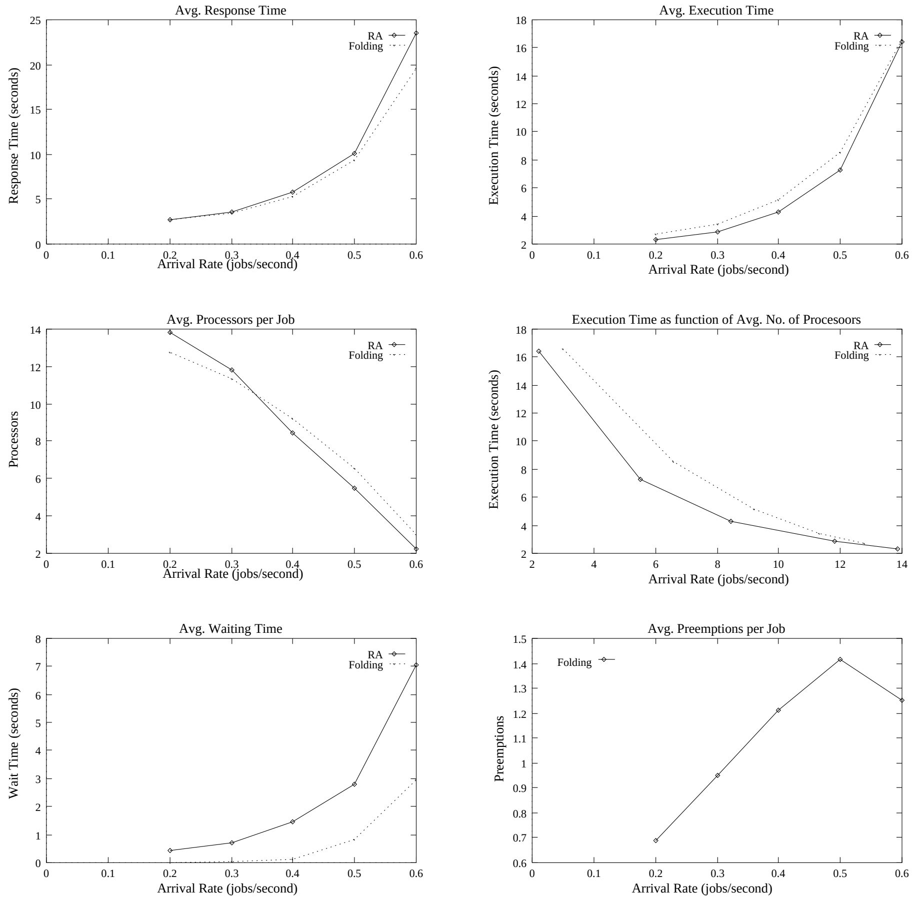

Fig. 4. RA versus Folding: Synthetic Workload with Constant Demand

Performance under Different Speedup Curves. Figure 6 compares the performance of the RA and the Folding policies when scheduling a CG workload with a 300x300 matrix. When preempted, approximately 0.72 megabytes of data require redistribution. The speedup curve of the workload is shown in Figure 2. The interarrival times are exponentially distributed, while the workload demand is constant.

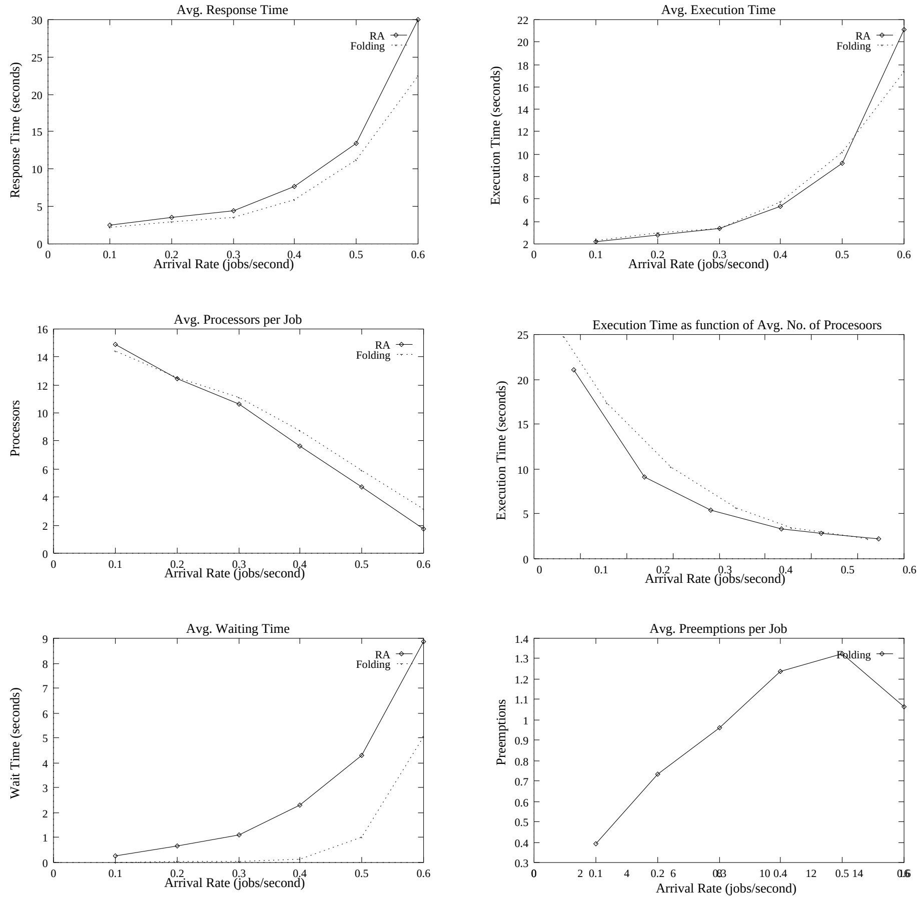

Fig. 5. RA versus Folding: Synthetic Workload with Exponential Demand

Once again, the Folding policy exhibits a better response time than RA for all arrival rates. The maximum improvement in the response time is approximately 19%. As in the case of synthetic workload, the execution time is better under RA policy than under Folding for all arrival rates. The maximum difference is approximately 20%. The average number of preemptions per job is small at both lower as well as higher arrival rates, while it peaks in the middle of the range. At lower arrival rates, an executing job is less likely to be preempted during its execution, since new jobs arrive at a slower rate. At higher arrival rates, several of the currently executing jobs are likely to have partitions consisting of single processors, and these can not be preempted for further folding. Thus, the average number of preemptions goes down at both higher and lower arrival rates.

Figure 7 shows the performance of the RA and the Folding policies when scheduling a CG workload with a 500x500 matrix. All other parameters are identical to the 300x300 case, except that a preemption requires redistribution of approximately 2 megabytes. The speedup curve is shown in Figure 2. It is seen from these results that as speedup curve becomes flatter (i.e. a 300x300 matrix as opposed to a 500x500 matrix), the improvement in the waiting time (and hence the response time) under Folding as compared to RA becomes more pronounced.

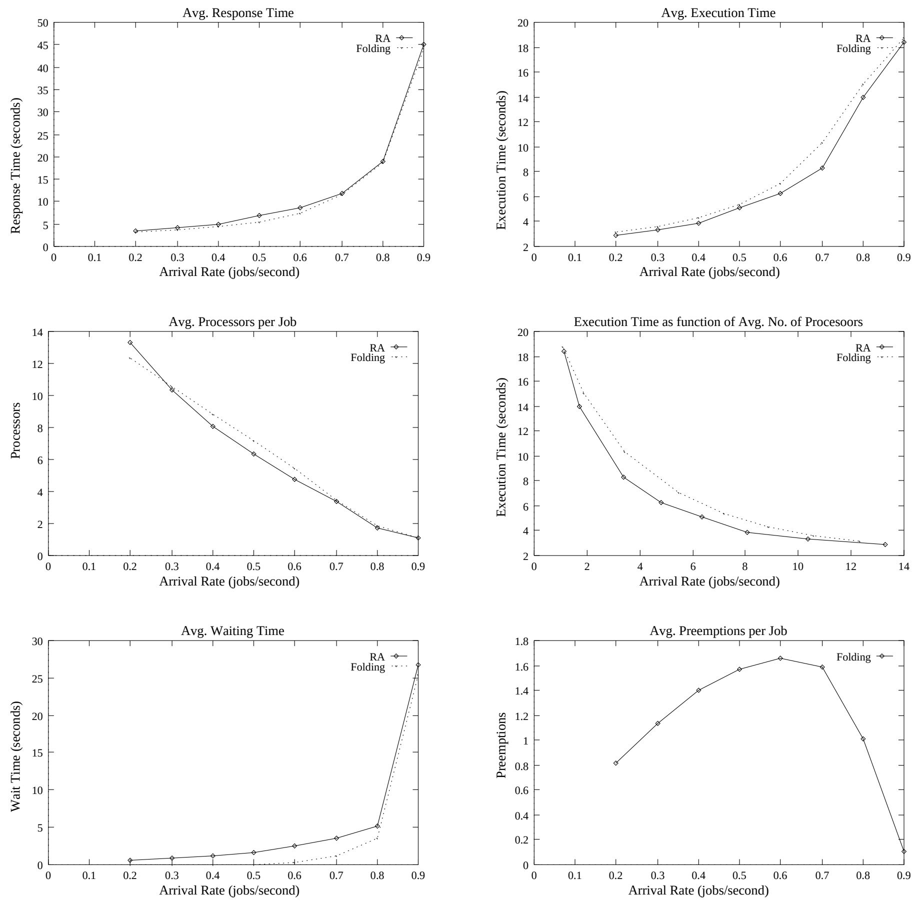

Fig. 6. RA versus Folding: Conjugate Gradient Workload with a 300x300 Matrix

Performance under Multiclass Workloads. Figure 8 compares the performance of the RA and the Folding policies when scheduling a two-class workload. Each class constitutes approximately 50% of the total workload. The first class (class A) consists of a CG program with a matrix size of 300x300, while the

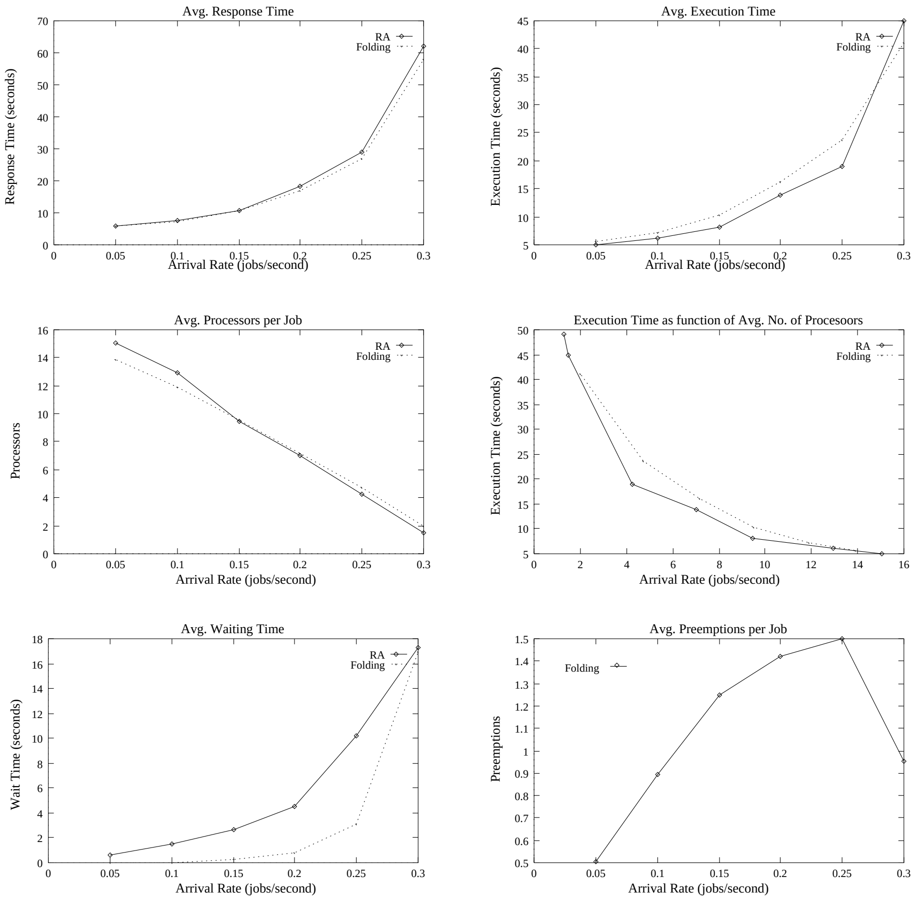

Fig. 7. RA versus Folding: Conjugate Gradient Workload with a 500x500 Matrix

second class (class B) has a matrix size of 500x500. Folding performs better than RA while scheduling a multiclass workload. Being a dynamic policy, Folding can adapt to transient changes in the workload flow better than RA. This behavior, as seen previously, is also observed when the workload demands are exponentially distributed rather than being constant. The maximum improvement in the response time is approximately 19%, achieved at relatively high arrival rates. Figure 9 compares the performance of the two policies when 20% of the programs are from class A while 80% come from class B. Since the workload now has less diversity, the difference between the performance of the two policies is narrowed. Specifically, the maximum improvement in the response time under Folding is approximately 9%, as opposed to 19% in the previous case.

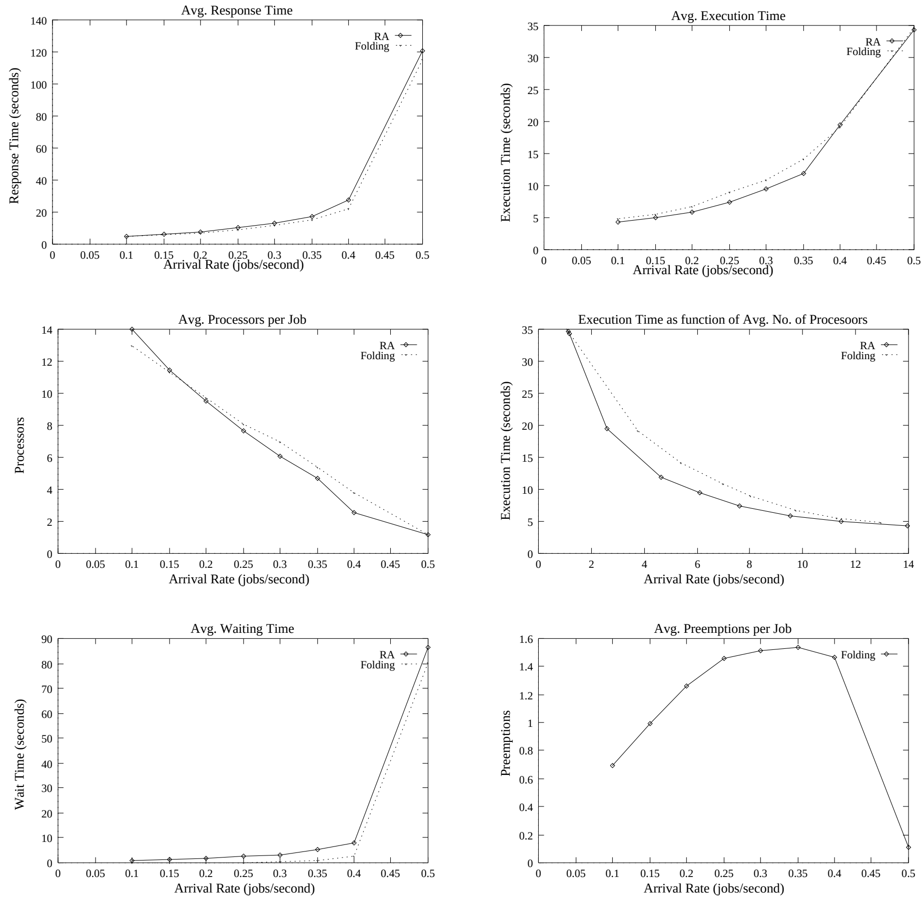

Fig. 8. RA versus Folding: Multiclass Workload 1

3.3 Comparison under Higher Numbers of Processors

The general behavior observed in the previous sections remains consistent as the number of processors in the multiprocessor increases. Figure 10 presents a comparison of the RA and the Folding policies with 500x500 CG workload. The workload demand is constant while the interarrival times have a negative exponential distribution. All the other parameters are the same as in the previous analysis (e.g. Section 3.2.2). It can be seen that better response times are obtained under Folding than under RA for most arrival rates. As the number of processors in a multiprocessor increases, good workload speedup characteristics are necessary to take advantage of dynamic scheduling provided under Folding. It is possible that under Folding for a job to be unfolded to run on the entire

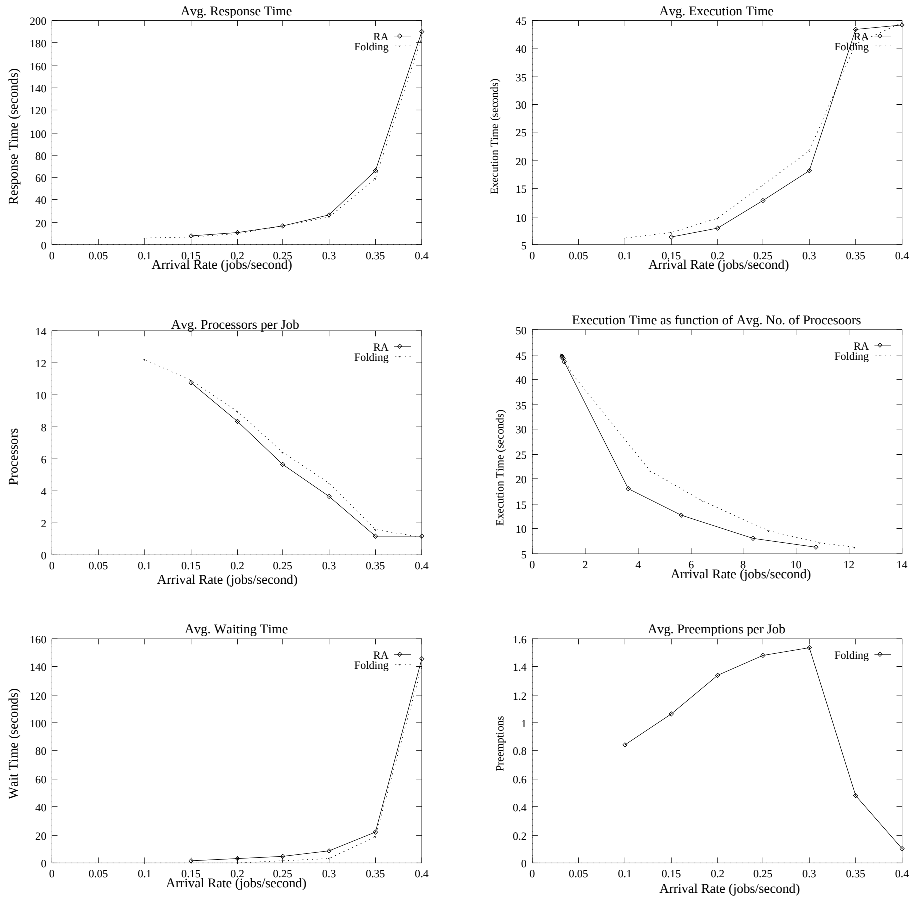

Fig. 9. RA versus Folding: Multiclass Workload 2

multiprocessor. If the job has poor speedup characteristics, or if the job has negative speedup (possible when allocated a very large number of processors), unfolding may prove detrimental to the overall performance.

4 Conclusions

This paper presents an experimental comparison between an adaptive scheduling policy (RA) and a dynamic scheduling policy (Folding). Both policies are implemented on a parallel computer with a message passing architecture. An open system model of the workload flow is implemented. The observed response time is the primary metric used for comparison. The results indicate that it is possible to achieve improved average response time using the dynamic Folding

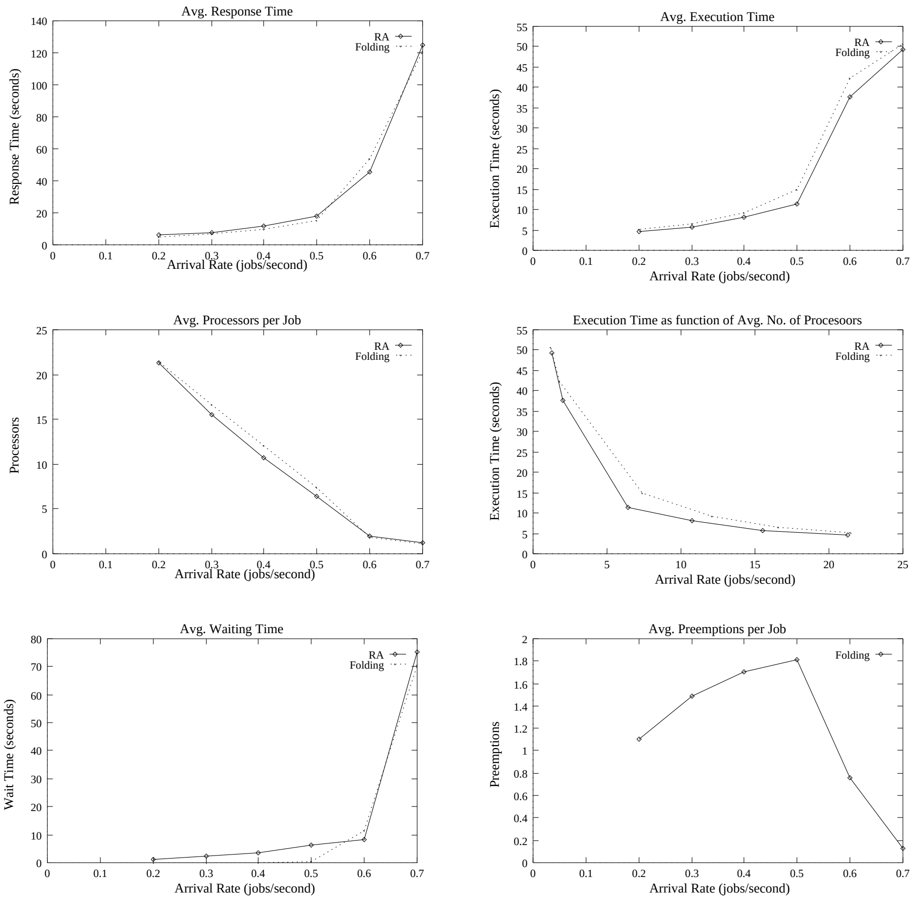

Fig. 10. RA versus Folding: 32 Processors

policy instead of an adaptive policy in many workload environments. In addition, the following conclusions can be drawn from the observed results:

- In most cases, the improvement in the response time is a direct result of the reduction in the time a job spends waiting in the ready queue.
- As jobs become less scalable, the advantages of using a dynamic scheduling policy like Folding become more apparent. The Folding policy is able to dynamically preempt processors from a currently executing job (where they are not providing any significant improvement in the execution time) and allocate them to a new job to take advantage of the efficient use of a few processors allocated to a newly arrived job. The adaptive policy may require the new arrivals to wait until at least one of the executing jobs completes.

- As the workload flow becomes more *random* (e.g. exponential demand as opposed to constant), Folding performs better than RA. The dynamic nature of the Folding policy allows it to adapt more quickly to transient changes common under exponentially distributed workload demands. The same is true in case of a multiclass workload.
- The overhead incurred by a workload while executing under a dynamic policy (e.g, Folding) can be relatively high. Careful implementation of the workload-scheduler interface and implementation of a conservative processor allocation policy can help reduce this overhead.
- Although dynamic policies can improve performance, excessive use of preemptions to reallocate processors can prove detrimental.

It is somewhat interesting to note that Folding and RA exhibit similar overall performance, typically within 10-15% of each other. The overall conclusion is that that a dynamic policy like Folding should be used to schedule jobs on a multiprocessor if the workload speedup characteristics are non-linear or if the demand is not constant (i.e, either exponentially distributed or multi-class workload) and if repartitioning overhead is low. An adaptive policy like RA should be used if the workload demand is constant, the workload is single-class, and has good speedup characteristics. An example is shown in Figure 11. The workload used for this comparison is similar to the synthetic workload described in Section 2, except that it has 15 phases instead of 250. This lowers the execution time relative to the scheduling overhead. In this case, RA outperforms Folding. It may also be possible for Equipartitioning to perform better than Folding under certain circumstances. Some pertinent results are presented in [IPS96]. Mary Vernon has suggested [V96] that on parallel systems of the future, the job arrival rates would be very low (30 jobs/hour) and at such low arrival rates, Folding and Equipartitioning may have similar performance.

Areas for future study include validating the results on a larger multiprocessor and comparing the policies under various distributions of arrival rates and workload demands. The experiments can be repeated on other machines using different processor interconnection networks and/or different message routing algorithms. The data from these experiments can be used to create better analytical and simulation models.

5 Acknowledgments

Evgenia Smirni, Emilia Rosti, Manish Madhukar and Jürgen Brehm provided invaluable help and suggestions throughout this study. We also wish to thank the staff members of the Center for Computational Sciences at the Oak Ridge National Labs for the use of the Intel Paragon. We also thank the referees for their helpful comments.

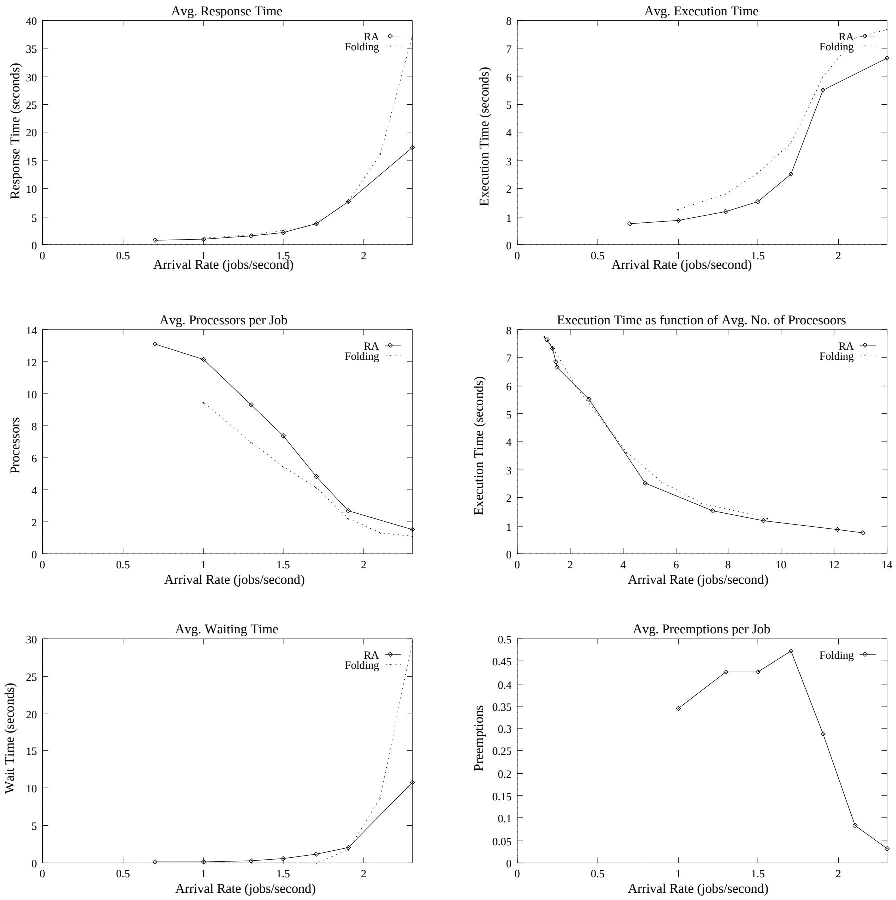

Fig. 11. RA versus Folding: Synthetic Workload with 15 Phases, 66000 Computations per Phase

References

[BMSD95] J. Brehm, M. Madhukar, E. Smirni, L. W. Dowdy, "PerPreT - A performance prediction tool for massively parallel systems," *Int. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation*, September 1995.

[CMV94] S.-H. Chiang, R.K. Mansharamani, M.K. Vernon, "Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies," *Proc. ACM SIGMETRICS*, 1994, pp. 33-44.

[DCDP90] K. Dussa, B.M. Carlson, L.W. Dowdy, K.-H. Park, "Dynamic partitioning in a transputer environment," *Proc. ACM SIGMETRICS*, 1990, pp. 203-213.

[GTU91] A. Gupta, A. Tucker, S. Urushibara, "The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications," *Proc. ACM SIGMETRICS*, 1991, pp. 120-132.
[INT93] Intel Corporation, **Paragon OSF/1 User's Guide**, 1993.
[IPS96] N. Islam, A. Prodormidis and M. Squillante, "Dynamic Partitioning in Different Distributed-Memory Environments," *In this volume.*
[LV90] S.T. Leutenegger, M.K. Vernon, "The performance of multiprogrammed multiprocessor scheduling policies," *Proc. ACM SIGMETRICS*, 1990, pp. 226-236.
[MEB88] S. Majumdar, D.L. Eager, R.B. Bunt, "Scheduling in multiprogrammed parallel systems," *Proc. ACM SIGMETRICS*, 1988, pp. 104-113.
[MVZ93] C. McCann, R. Vaswani, J. Zahorjan, "A dynamic processor allocation policy for multiprogrammed shared memory multiprocessors," *ACM Transactions on Computer Systems*, Vol 11(2), February 1993, pp. 146-178.
[MZ94] C. McCann, J. Zahorjan, "Processor allocation policies for message-passing parallel computers," *Proc. ACM SIGMETRICS*, 1994, pp. 19-32.
[P96] J. Padhye, "Preemptive versus non-preemptive processor allocation policies: an empirical comparison", Technical Report, Department of Computer Science, Vanderbilt University, 1996.
[PD89] K.-H. Park, L.W. Dowdy, "Dynamic partitioning of multiprocessor systems," *International Journal of Parallel Programming,* Vol 18(2), 1989, pp. 91-120.
[RSDSC94] E. Rosti, E. Smirni, L.W. Dowdy, G. Serazzi, B.M. Carlson, "Robust partitioning policies for multiprocessor systems," *Performance Evaluation*, Vol 19(2-3), March 1994, pp. 141-165.
[SEV94] K.C. Sevcik, "Application scheduling and processor allocation in multiprogrammed multiprocessors," *Performance Evaluation*, Vol 19(2-3), March 1994, pp. 107-140.
[SRSDS94] E. Smirni, E. Rosti, G. Serazzi, L. W. Dowdy, K. C. Sevcik "Performance gains from leaving idle processors in multiprocessor systems" *Proc. International Conference on Parallel Processing*, 1995.
[SST93] S.K. Setia, M.S. Squillante, S.K. Tripathi, "Processor scheduling in multiprogrammed, distributed memory parallel computers," *Proc. ACM SIGMETRICS*, 1993, pp. 158-170.
[TG89] A. Tucker, A. Gupta, "Process control and scheduling issues for multiprogrammed shared-memory multiprocessors," *Proc. of the 12th ACM Symposium on Operating Systems Principles*, 1989, pp. 159-166.
[V96] Mary Vernon, Personal Communication.
[ZB91] S. Zhou, T. Brecht, "Processor pool-based scheduling for large-scale NUMA multiprocessors," *Proc. ACM SIGMETRICS*, 1991, pp. 133-142.
[ZM90] J. Zahorjan, C. McCann, "Processor scheduling in shared memory multiprocessors," *Proc. ACM SIGMETRICS*, 1990, pp. 214-225.

This article was processed using the LaTeX macro package with LLNCS style