Selective Buddy Allocation for Scheduling Parallel Jobs on Clusters *

Vijay Subramani† Rajkumar Kettimuthu† Srividya Srinivasan† Jeanette Johnston‡
P. Sadayappan†

Abstract

In this paper, we evaluate the performance implications of using a buddy scheme for contiguous node allocation, in conjunction with a backfilling job scheduler for clusters. When a contiguous node allocation strategy is used, there is a trade-off between improved run-time of jobs (due to reduced link contention and lower communication overhead) and increased wait-time of jobs (due to external fragmentation of the processor system). Using trace-based simulation, a buddy strategy for contiguous node allocation is shown to be unattractive compared to the standard non-contiguous allocation strategy used in all production job schedulers. A simple but effective scheme for selective buddy allocation is then proposed, that is shown to perform better than non-contiguous allocation.

1 Introduction

Job schedulers for clusters and parallel computer systems generally do not take system topology and node proximity considerations into account when making scheduling decisions for parallel jobs [6, 8, 20]. The available processors are generally maintained in a free-list - they are removed from the free-list for allocation to new jobs, and added back to the free-list when jobs complete and release their processors. Currently, the allocation of nodes to jobs on the Cplant system at Sandia National Laboratories [18] also does not take node proximity into account when allocating nodes to parallel jobs. However there is evidence that for some communication-intensive parallel programs, a contiguous node allocation can result in up to a 40% reduction in execution time. Thus it is of interest to consider the implications of contiguous node allocation on job scheduling.

There is a fundamental trade-off involved when a contiguous node allocation strategy is used for scheduling of parallel jobs instead of the current practice of arbitrary non-contiguous node allocation in production job schedulers: The run-times of jobs can be expected to decrease when a contiguous node allocation strategy is used, due to decreased link contention between independent concurrently executing jobs in the system. The extent of improvement can be quite different for the jobs in the mix - communication-intensive jobs will benefit the most, while parallel jobs with very little communication will see no noticeable reduction in execution time. The wait-times for jobs can be expected to increase, due to external fragmentation. External fragmentation exists when a sufficient number of processors are available to satisfy a request, but they cannot be allocated contiguously. With some contiguous allocation strategies that restrict the sizes of allocated processor groups, there may also be wasted processor cycles due to internal fragmentation (when more processors are allocated to a job than it requests).

The effectiveness of a contiguous node allocation strategy is thus clearly dependent on which of the above effects is more dominant. Since the two goals of reduced runtime and reduced wait-time are conflicting, our challenge is to design a strategy that achieves a reduction in overall turn-around time. There has been considerable prior research into each of the two topics of Scheduling of parallel jobs [5, 17] and contiguous node allocation strategies [1, 2, 3, 9, 12, 21].

There have also been a few studies that have considered both these issues in combination [10, 14, 16]. However, only [14] addresses the impact of contiguous node allocation schemes in conjunction with a job scheduling policy that takes fairness into consideration - by use of a FCFS (First Come First Served) scheduling policy.

In [14], contiguous and non-contiguous node allocation schemes for mesh-connected systems were evaluated in conjunction with a job scheduler. Both simulation-based evaluation and experimental measurements on an Intel Paragon parallel system were performed. The simulation-based evaluation first involved a fragmentation study, where synthetic job streams with different distributions were evaluated at different simulated loads. Here, communication costs were not modeled. The conclusion was that contiguous allocation schemes suffered considerably from

*Supported in part by a grant from Sandia National Laboratories
†Dept. of Computer and Info. Science, Ohio State University, Columbus, OH 43210, {subraman,kettimut,srinivas,saday}@cis.ohio-state.edu
‡MS 1110, Sandia National Laboratories, Albuquerque, NM 87185-1110, jjohnst@mp.sandia.gov

external fragmentation and consequent poor system utilization. Then, simulation-based message-passing experiments were performed, where the simulator was extended to model communication costs and contention during interprocessor communication. Programs with different communication patterns were evaluated, including some of the NAS benchmarks. The conclusion was that a multiple-buddy strategy was best, performing better than fully contiguous allocation schemes and random non-contiguous allocation. Finally, experimental evaluation on a parallel Intel Paragon system was performed. Compared to the simulation-based results, the experimental measurements on the Paragon system did not reveal much benefit from using a partially contiguous allocation.

In this paper, we address the same issues as in [14]. However, there are some important differences: Their study did not vary the communication parameters, but instead used fixed parameters corresponding to a single system. Since the effectiveness of contiguous allocation will depend heavily on the run-time improvement, it is of interest to perform evaluations under different assumptions about the communication overhead. The previous study treated all jobs equivalently, with respect to the node allocation strategy. As we show later, the effectiveness of contiguous node allocation is greatly enhanced by applying it selectively to jobs.

Using data from a job trace collected at the Cornell Theory Center [4], we demonstrate that a selective buddy allocation scheme used in conjunction with EASY backfilling [13, 17, 19] is consistently superior to the traditional non-contiguous node allocation strategy. The rest of the paper is organized as follows. In Section 2, we provide some background and motivation for this work. In Section 3, we evaluate a buddy node allocation policy and discuss its shortcomings. A selective buddy allocation scheme is proposed in Section 4, and evaluated first under the assumption of perfect user estimation of job run-times. In Section 5, we present simulation data using actual user estimates of job runtime. Related work is discussed in Section 6 and conclusions are provided in Section 7.

2 Background and Motivation

Scheduling is usually viewed using a 2D chart with time along the horizontal axis and the number of processors along the vertical axis. Each job can be thought of as a rectangle whose length is the user estimated run time and width is the number of processors required. The simplest way to schedule jobs fairly is to use the First-Come-First-Served (FCFS) policy. This approach suffers from low system utilization. Backfilling [5, 17] was proposed to improve the system utilization and has been implemented in several production schedulers [8]. Backfilling works by identifying holes in the 2D chart and moving forward smaller jobs that fit those holes. There are two common variations to backfilling - conservative and aggressive. In conservative backfilling, a smaller job is moved forward in the queue as long as it does not delay any previously queued job. In aggressive backfilling, also known as EASY backfilling [13, 19], a small job is allowed to leap forward as long as it does not delay the job at the head of the queue. Backfilling thus allows significant improvement in system utilization without sacrificing fairness.

The common metrics used to evaluate the performance of scheduling schemes are the average turnaround time and the average bounded slowdown [17]. We use these metrics for our studies. The bounded slowdown of a job is defined as follows:

Bounded Slowdown = (Wait time + Max(Run time, 10))/ Max(Run time, 10)

The threshold of 10 seconds is used to limit the influence of very short jobs on the metric.

For communication intensive (CI) jobs, network contention is a significant bottleneck. If the nodes allocated to a CI job are not contiguous (i.e., the nodes are on different switches) and if the hop distance is relatively high, then the messages exchanged between those nodes have to pass through several links and may have to contend with the messages from other CI jobs. Instead, if we allocate the job to nodes on the same switch or to nodes with a low hop distance between them, network contention can be reduced, thereby improving the runtime for those jobs. We performed some experiments using the NAS FFT parallel benchmark to evaluate the amount of benefit that can be obtained from contiguous node allocation when compared to random node allocation. The experiments were performed on the toroidal mesh-connected Cplant system at Sandia National Laboratory [18].

	Time Taken	
Set	Contiguous	Non Contiguous
1	0:01:37	0:02:38
2	0:01:33	0:02:40
3	0:01:35	0:02:35
4	0:01:33	0:02:42
5	0:01:36	0:02:39
6	0:01:36	0:02:38
7	0:01:33	0:02:39
8	0:01:33	0:02:39

Table 1. Execution time for FFT with Contiguous vs. Non-contiguous node allocation.

Eight sets of 16 node FFT's were run concurrently on a 128 node mesh connected system. The table above shows the time taken for the FFT application with contiguous and non-contiguous node allocation. For the contiguous node allocation, each job was assigned contiguous nodes on a mesh row. For the non-contiguous node allocation, the nodes were assigned randomly. It can be observed that more than 40% improvement in runtime can be obtained with contiguous node allocation. But the percentage improvement in runtime is very dependent on the nature of the application. So, we study the impact of contiguous node allocation assuming different percentages of improvement in job runtime.

From the collection of workload logs available from Feitelson's archive [4], the CTC workload trace was used to evaluate the proposed schemes. This trace was generated by a 430-processor system (a subset of jobs spanning a period of one month from the entire trace of jobs was used for the experiments, in order to reduce the total simulation time required). The trace contains the actual execution time as well as the wall-clock limit requested by the user for each job. Under normal load, with aggressive backfilling, using FCFS as the scheduling priority, the utilization was 51 percent. Although schedulers in practice can only make scheduling decisions based on the wall-clock limit provided by the user, we first carry out our simulation experiments using the actual job runtime as the basis for decisions. This to get a clear understanding of the issues under investigation, without any effects that may be a consequence of inaccuracy of estimates by the user. Later in Section 5, we conduct simulation experiments where the user-specified wall-clock limit is used as the basis for scheduling decisions (but the actual recorded job runtime is used to determine when the simulated job execution terminates, so that the simulation can model what would happen in reality)

3 Evaluation of Buddy Node Allocation

As discussed earlier, the effectiveness of a contiguous node allocation scheme in a space-shared cluster environment will depend on the extent of improvement of job run-times when they are allocated to contiguous processors rather than to arbitrary processors in the system. If the improvement in runtime is relatively small, the increase in job wait-times can be expected to exceed the reduction in execution time, rendering contiguous node allocation unattractive. On the other hand, if the reduction in job execution time were very large, the overall turn-around time would decrease despite an increase in wait time.

In this section, we evaluate the effectiveness of a buddy scheme for contiguous node allocation, under different assumptions about the extent of run-time improvement from contiguous node allocation. We first simulate the scheduling and execution of jobs in the CTC trace using the default non-contiguous node allocation strategy, and EASY Backfilling. The free processors of the system are placed in a linked list. When a new job is scheduled, the needed number of processors are removed from the free list and allocated to the job. When a job completes execution, the processors are freed and inserted into the free list. Thus no consideration is given to the relative physical proximity of processors when node allocation is performed.

Next, we simulate the scheduling and execution of jobs using a buddy scheme for node allocation. A brief description of the scheme is provided below.

The list of buddies is maintained as a binary tree in which the size of the root buddy is equal to size of the system. The sum of the sizes of its two child buddies is equal to its size. All the leaf buddies are of size one. Each buddy contains a list of time intervals in which it is free.

Nodes are allocated to the jobs in the following manner

- The smallest buddy is chosen such that the number of nodes in the buddy is at least equal to the number of nodes requested by the job.

- If there are many choices, one with the earliest start time is chosen.

- In case of a tie, a buddy is chosen with the smallest value for 'largest buddy of which it is a part of' (to reduce external fragmentation).

- The free time list for the selected buddy and all its children and parent buddies are updated.

Simulations were performed under different assumptions about the improvement in run-time with contiguous allocation. For example, if the improvement factor is 10%, the execution time of the job under a contiguous node allocation is made 10% less than the execution time with the arbitrary non-contiguous allocation.

Fig. 1 indicates that the Buddy allocation scheme results in a decrease in the average slowdown only when a 50% improvement in runtime is assumed for contiguous allocation. Fig. 2 indicates that the Buddy allocation scheme results in a decrease in the average turnaround time only if there is 25% or higher improvement in runtime due to contiguous allocation.

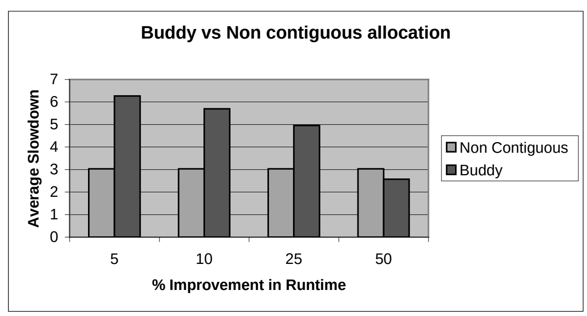

Figure 1. Comparison of average slowdowns of jobs under the Buddy allocation scheme and-Non contiguous allocation scheme.

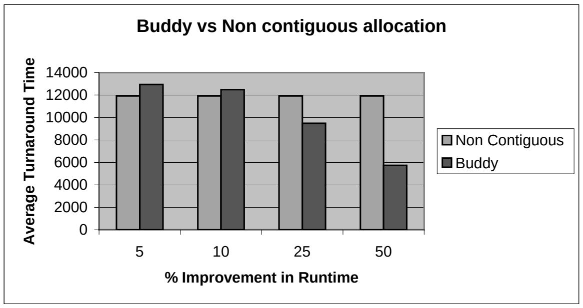

Figure 2. Comparison of average turnaround times of jobs under the Buddy allocation scheme and Non-contiguous allocation scheme.

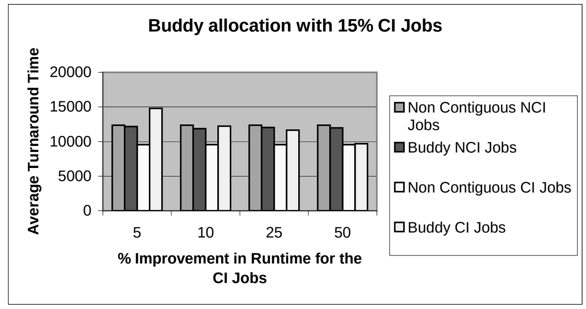

Figure 4. Comparison of the average turnaround times of the CI and NCI jobs under the Buddy allocation scheme and Non-contiguous allocation scheme.

In practice, it is extremely unlikely that all jobs in the system are communication intensive and thereby able to benefit from contiguous node allocation. So we next carried out a simulation where we assumed that only 15% of the jobs were communication intensive. In this case, only the communication intensive jobs were allocated contiguous nodes, while the other jobs were allocated nodes without requiring them to be contiguous.

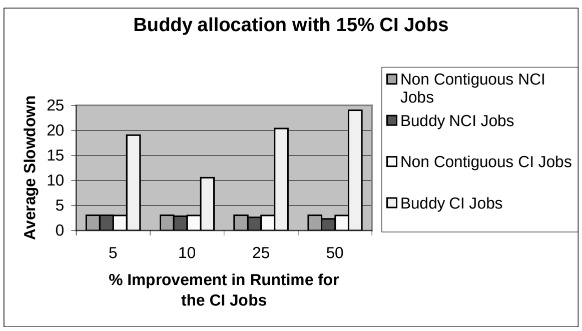

Figure 3. Comparison of the average slowdowns of the Communication Intensive (CI) and Non-communication Intensive (NCI) jobs under the Buddy allocation scheme and Non-contiguous allocation scheme.

Fig. 3 and 4 show that the performance of the communication intensive jobs has deteriorated considerably, when compared to Fig. 1 and 2. The reason for this became clear when we examined the simulation data more closely. When all jobs are considered communication-intensive (results in Fig. 1 and 2), there is effectively a decrease in the offered load (total utilized processor-seconds for all jobs combined) when contiguous allocation is used. When only 15% of the jobs are communication-intensive, there is much less decrease in the offered load since only a small fraction of the jobs now have reduced running times. From Fig. 3 and 4, the prospects for contiguous node allocation schemes do not look very good when the fraction of communication intensive jobs is low. However, as we explain in the next section, by using a selective strategy for contiguous node allocation, it is possible to significantly improve performance.

4 A Selective Buddy Allocation Strategy

An examination of the turnaround times of jobs with the buddy allocation strategy revealed that even when the average slowdown was worse than the non-contiguous case, there were indeed several jobs whose slowdown improved considerably. In general, long jobs had lower average slowdown and tended to show improvement, while short jobs had higher slowdown and got worse under contiguous allocation. In retrospect, this was to be expected. Those jobs that had a relatively high slowdown under non-contiguous allocation had very little likelihood of improving with a contiguous allocation. This is because their turnaround time is dominated by the wait-time. Thus, with a contiguous node allocation, there is very limited potential for a decrease in turnaround time through decrease in run-time, and it is unlikely to compensate for the increased wait time. In contrast, jobs that have relatively low (close to 1.0) slowdown under non-contiguous allocation have their turnaround time dominated by execution time; with a contiguous allocation, these jobs have much better prospects for decreased turnaround time.

These observations prompted consideration of a selective buddy allocation strategy: use contiguous allocation selectively, only for those jobs whose expected turnaround time with contiguous allocation is lower than with non-contiguous allocation. When a job is to be scheduled, the two possibilities are both evaluated - contiguous allocation, with a longer expected wait time but lower run time; and non-contiguous allocation, with lower wait time, but longer run time. A choice of contiguous or non-contiguous alloca-

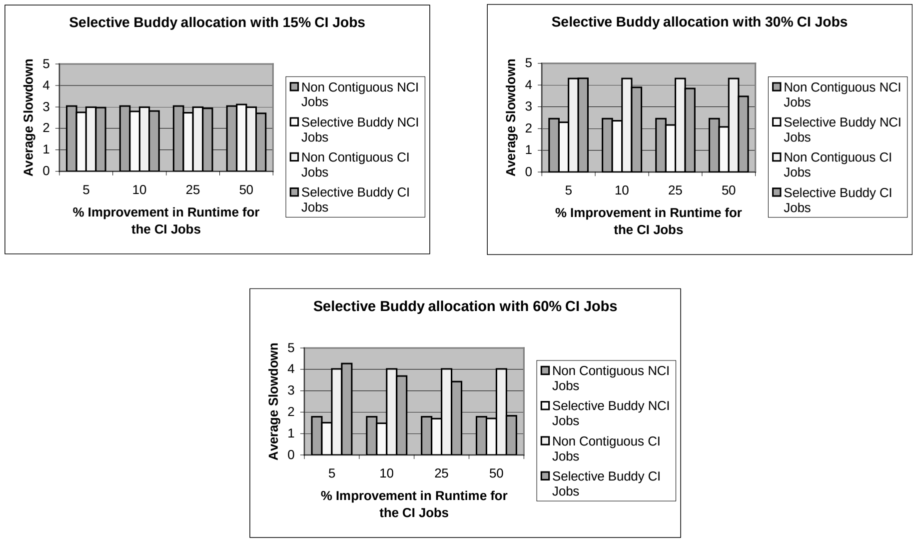

Figure 5. Comparison of the average slowdowns of the CI and NCI jobs under the Selective Buddy allocation scheme and Non-contiguous allocation scheme.

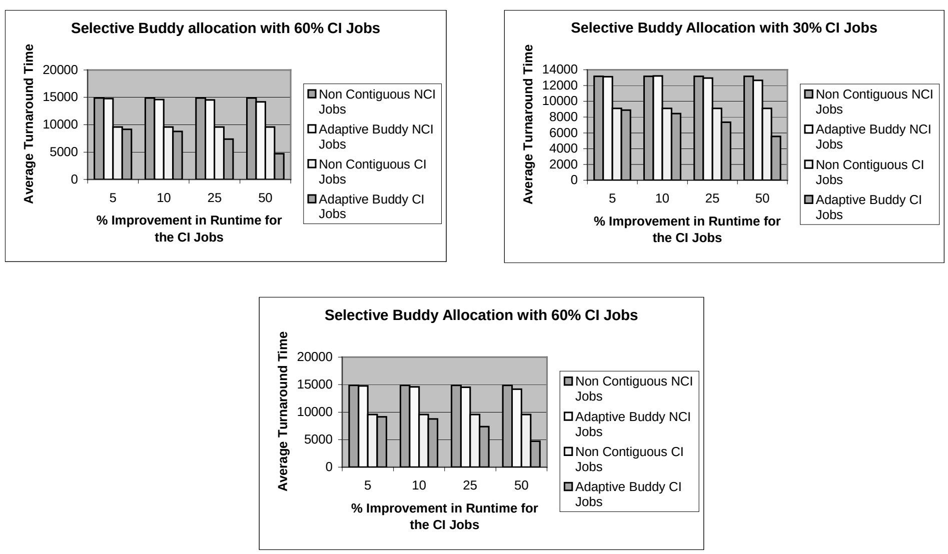

Figure 6. Comparison of the average turnaround times of the CI and NCI jobs under the Selective Buddy allocation scheme and Non-contiguous allocation scheme.

tion is made dynamically, based on estimated completion time with the two choices. The simulation results below show that this simple selective strategy is effective, even when the fraction of CI jobs is relatively small.

Fig. 5 shows the average bounded slowdown for the jobs under the selective buddy scheme. When the trace contains 15% CI jobs, the selective scheme reduces the slowdown of both the CI and NCI jobs. The percent improvement in slowdown for the CI jobs increases as the percentage improvement in runtime for those jobs is increased. Similar trends are observed with 30% and 60% of jobs being communication intensive jobs.

Fig. 6 shows the turnaround time for the jobs under this scheme. It can be observed that this scheme improves the turnaround time of the CI jobs without adversely affecting the NCI jobs. It also improves the turnaround time of the NCI jobs as the percentage improvement in the runtime increases. This is because the reduction in runtime for the CI jobs also has the side-effect of reducing the overall load in the system, thereby benefitting the NCI jobs too.

Thus the selective buddy scheme outperforms the non contiguous allocation scheme even for moderate amounts of improvement in runtime and even when only a small percentage of the jobs are communication intensive.

5 Evaluation with User Estimates of Job Run-time

We have so far assumed that the user estimates of runtime are perfect. Now, we consider the effect of user estimate inaccuracy on the Selective Buddy Allocation scheme. Fig. 7 shows that for 15% CI jobs, as we increase the percentage improvement in runtime for the CI jobs, the average turnaround time for these jobs decreases with the selective buddy scheme. But from Fig. 8 it can be observed that slowdown for the CI jobs is worse with the selective scheme even with higher percentage improvement in runtime.

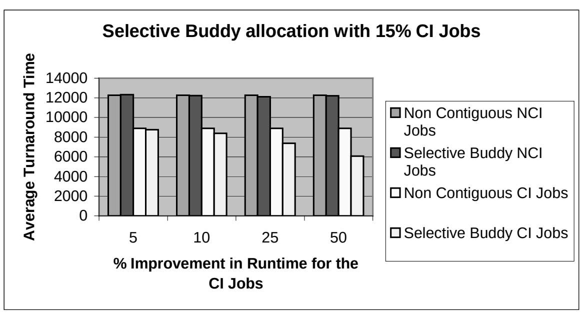

Figure 7. Comparison of the average turnaround times of the CI and NCI jobs under the Selective Buddy allocation scheme and Non contiguous allocation scheme.

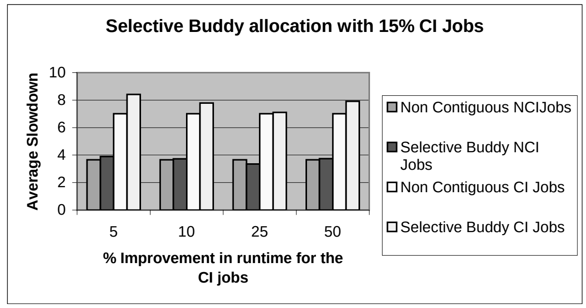

Figure 8. Comparison of the average slowdowns of the CI and NCI jobs under the Selective Buddy allocation scheme and Non contiguous allocation scheme.

We believe that this is due to abnormally aborted jobs and the jobs with higher over-estimation factor, which tend to excessively skew the average slowdown of jobs in a workload. Consider a job requesting a wall-clock limit of 24 hours, that is queued for 1 hour, and then aborts within one minute due to some fatal exception. The slowdown of this job would be computed to be 60, whereas the average slowdown of normally completing long jobs is typically under 2. If even 5% of the jobs have a high slowdown of 60, while 95% of the normally completing jobs have a slowdown of 2, the average slowdown over all jobs would be around 5. In order to verify our conjecture that the worse slowdown for CI jobs with the selective buddy scheme is due to poorly estimated jobs, we group the jobs into two different estimation categories. One category contains jobs that are well-estimated (the estimated time is not more than twice the actual runtime of that job) and badly estimated jobs (the estimated runtime is more than twice the actual runtime).

From the Fig. 9 and 10 it can be observed that the selective buddy scheme improves turnaround times of the CI jobs in both the categories (good and bad estimators). Fig. 9 show that the average slowdown for the CI jobs in the good estimators category improves with the selective buddy scheme and trends are very similar to that observed in the previous section with simulations using the actual run-time as estimate.

For the poorly estimated jobs, the average slowdown for CI jobs worsens with the selective buddy scheme. This is due to the fact that these jobs are badly estimated i.e, these jobs finally end up as short jobs, but they are considered as long jobs by the scheduler implementing the selective buddy scheme. These jobs are made to wait to get a better turnaround time via a contiguous allocation, but they complete much earlier than expected. Thus their longer wait to get a contiguous allocation does not pay off since their runtime is much less than predicted. This effect of the se-

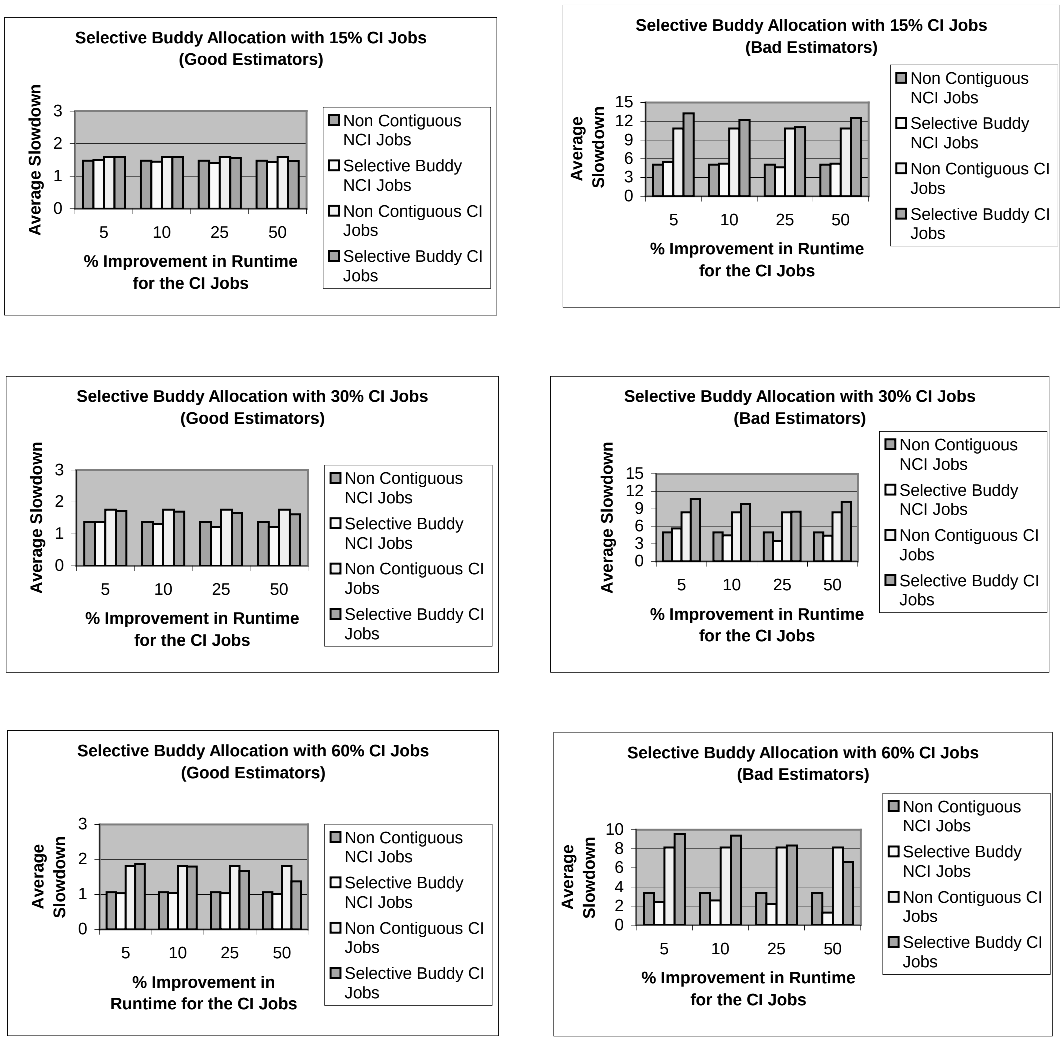

Figure 9. Comparison of the average slowdowns of the well estimated and the poorly estimated jobs (CI and NCI) under the Selective Buddy allocation scheme and Non-contiguous allocation scheme.

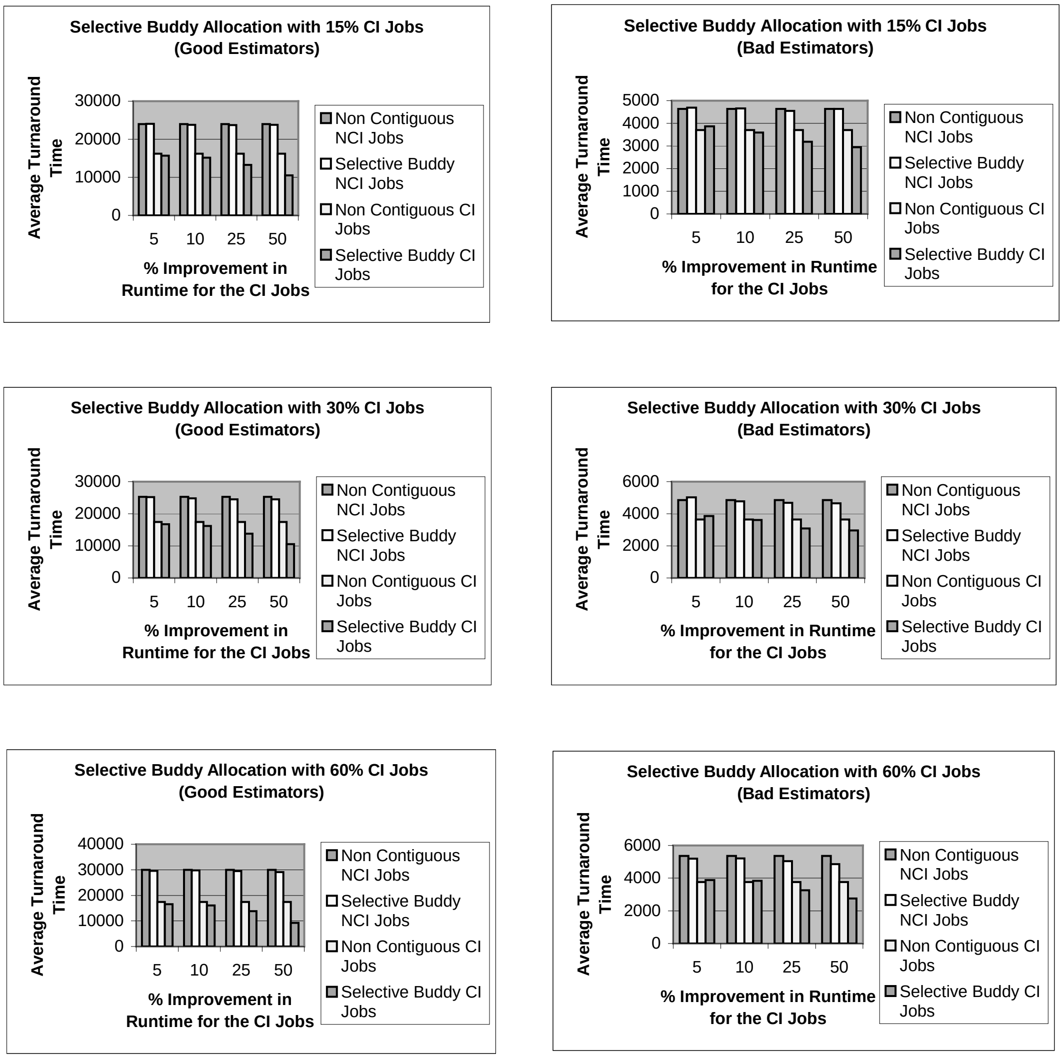

Figure 10. Comparison of the average turnaround times of the well estimated and the poorly estimated jobs (CI and NCI) under the Selective Buddy allocation scheme and Non-contiguous allocation scheme.

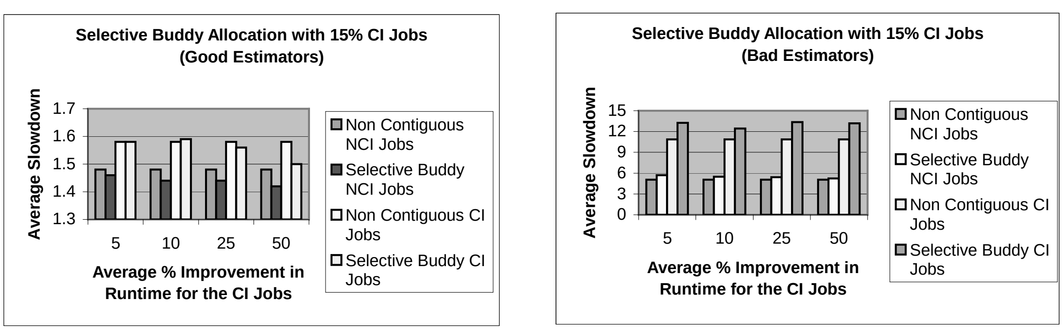

Figure 11. Comparison of the average slowdowns of the well estimated and the badly estimated jobs (CI and NCI) under the Selective Buddy allocation scheme and Non contiguous allocation scheme assuming a non-uniform percentage improvement for the CI jobs.

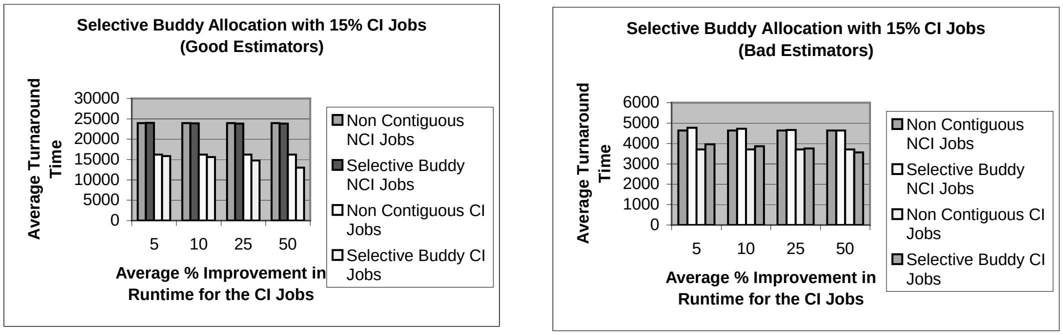

Figure 12. Comparison of the Average Turnaround Time of the Well and Badly Estimated Jobs (CI and NCI) under the Selective Buddy Allocation and the Non Contiguous Allocation Schemes assuming non-uniform percentage improvement for CI Jobs.

lective buddy allocation scheme may actually be desirable - it would encourage users to better estimate their job execution time, since poorly estimated jobs tend to suffer with respect to their wait-time!

5.1 Variable Percentage Improvement

So far we have assumed uniform percentage improvement in runtime for all the CI Jobs. But that assumption is not realistic so we now study the impact of the selective buddy allocation strategy assuming non-uniform percentage improvement in runtime for the CI Jobs i.e, given a percentage improvement X, we randomly assign the percentage improvement in runtime for the CI jobs in the range between 0 and 2X.

From Fig. 11 it can be observed that the selective buddy scheme improves the slowdowns of both the CI and NCI jobs for the good estimators. It can also be observed that the poorly estimated jobs slightly suffer. A similar trend is observed for the average turnaround time. This can be observed from Fig. 12. Thus, the Selective Buddy Allocation strategy outperforms the non contiguous allocation even when the CI jobs have variable improvements in their run-times.

6 Related Work

Several algorithms have been proposed previously for contiguous node allocation on hypercube and mesh computers. The Buddy strategy has been widely studied in the past [1, 3, 9]. Several studies [2, 12, 21] have focused on node allocation strategies for mesh connected systems. A fragmentation free allocation strategy for mesh connected parallel computers that minimizes message passing contention is presented in [7]. [15] presents a strategy that minimizes network contention due to both communication and I/O traffic. All the above studies focus exclusively on the topic of contiguous node allocation schemes, but do not address the issue of job scheduling onto the parallel systems.

The studies in [10, 16] focus on job scheduling techniques for improving the performance of hypercube computers. In [10], the roles of processor allocation and job scheduling in improving the performance of hypercube computers are compared. A new scheduling algorithm is proposed for improving the average response time of jobs and for improving the system utilization. A primary conclusion of the paper is that job scheduling considerations play a much more significant role in determining performance than the particular contiguous allocation strategy used. Further, it was concluded that there is very little advantage to using very sophisticated contiguous node allocation schemes instead of a very simple contiguous allocation scheme. In [16], a scheduling algorithm is presented for hypercube systems which provides greater improvements compared to the Scan algorithm of [10].

The scheduling algorithms in [10, 16] do not incorporate any fairness considerations. They would permit a later arriving job to be started at a time that it would be feasible to start an earlier arriving job, if it improved system utilization. Fairness is a highly desirable consideration in practical scheduling algorithms. For this reason, most production schedulers use some variation of backfilling along with a FCFS priority scheme. In this work, we consider the issue of contiguous node allocation in conjunction with a standard backfilling job scheduler, so that fairness considerations remain unchanged.

The approach considered in this paper either allocates a fully contiguous partition for a job, or an arbitrary collection of nodes (if the greater wait-time to acquire a fully contiguous partition exceeds the expected reduction in runtime). It may be of interest to consider allocations in between these two extremes, i.e. node allocations that are not fully contiguous, but nevertheless exhibit a high degree of proximity. Such as approach to node allocation has been pursued by Leung et. al. [11].

7 Conclusions

In this paper, we have addressed the issue of contiguous allocation in conjunction with a backfilling job scheduler. The relative performance (using average job slowdown and average turnaround time of jobs) for contiguous versus non-contiguous node allocation was characterized for different scenarios. A simple but effective selective buddy scheme was proposed, and demonstrated to have good performance relative to the standard non-contiguous node allocation strategy.

References

[1] M. Chen and K. G. Shin. Processor allocation in an n-CUBE multiprocessor using gray codes. *IEEE Transaction on Computers*, C(36):1396–1407, 1987.

[2] P. Chuang and N. Tseng. An efficient submesh allocation strategy for mesh computer systems. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 256–263, 1991.

[3] S. Dutt and J. P. Hayes. Subcube allocation in hypercube computers. *IEEE Transaction on Computers*, 40(3):341–352, 1991.

[4] D. G. Feitelson. Logs of real parallel workloads from production systems. http://www.cs.huji.ac.illogs/parallel/workload/logs.html.

[5] D. G. Feitelson, Rudolph, U. Schwiegelshohn, K. C. Sevcik, and Wong. Theory and practice in parallel job scheduling. In *Feitelson & Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing: IPPS '95 Workshop, Springer LNCS 949*. 1997.

[6] R. Henderson and D. Tweten. Portable Batch System: External reference specification. Technical report, NASA Ames Research Center, 1996.

[7] V. L. J. Mache and K. Windisch. Minimizing message-passing contention in fragmentation-free processor allocation. In *Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems*, pages 120–124, 1997.

[8] D. Jackson, Q. Snell, and M. J. Clement. Core algorithms of the maui scheduler. In *JSSPP*, pages 87–102, 2001.

[9] J. Kim, C. R. Das, and W. Lin. A top-down processor allocation scheme for hypercube computers. In *IEEE Transactions on Parallel and Distributed Systems*, volume 2, pages 20–30, 1991.

[10] P. Krueger, T. Lai, and V. Dixit-Radiya. Job scheduling is more important than processor allocation for hypercube computers. In *IEEE Transactions on Parallel and Distributed Systems*, volume 5, pages 488–497, 1994.

[11] V. Leung, E. Arkin, M. Bender, D. Bundee, J. Johnston, A. Lal, J. Mitchell, C. Phillips, and S. Seiden. Processor allocation on cplant: Achieving general processor locality using one-dimensional allocation strategies. In *Proceedings of IEEE Intl. Conf. on Cluster Computing*, 2002.

[12] K. Li and K. H. Cheng. A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system. *Journal of Parallel and Distributed Computing*, 12:79–83, 1991.

[13] D. Lifka. The ANL/IBM SP scheduling system. In *JSSPP*, pages 295–303, 1995.

[14] V. Lo, K. J. Windisch, W. Liu, and B. Nitzberg. Non-contiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 8(7):712–726, 1997.

[15] J. Mache, V. Lo, and S. Garg. Job scheduling that minimizes network contention due to both communication and i/o. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, 2000.

[16] P. Mohapatra, C. Yu, C. R. Das, and J. Kim. A lazy scheduling scheme for improving hypercube performance. In *Proceedings of the 1993 International Conference on Parallel Processing*, volume I - Architecture, pages I–110–I–117. CRC Press, 1993.

[17] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. In *IEEE Transactions on Parallel and Distributed Computing*, volume 12, pages 529–543, 2001.

[18] R. Riesen, R. Brightwell, L. A. Fisk, T. Hudson, J. Otto, and A. B. Maccabe. Cplant. In *Proceedings of the Second Extreme Linux Workshop at the 1999 USENIX Annual Technical Conference*, 1999.

[19] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY - loadleveler API project. In *JSSPP*, pages 41–47, 1996.

[20] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems, 1992.

[21] Y. Zhu. Efficient processor allocation strategies for mesh-connected parallel computers. *Journal of Parallel and Distributed Computing*, 16:328–337, 1992.