A Modified Adaptive Space-Sharing Policy for Non-dedicated Heterogeneous Cluster Systems

Amit Chhabra and Gurvinder Singh,

Department of Computer Science & Engineering Guru Nanak Dev University, Amritsar

Summary
Adaptive space-sharing algorithms are commonly used to realize the fullest strengths of dedicated homogeneous cluster computing systems since they produce better performance than fixed and variable-space sharing policies. However cluster of computers tend to be heterogeneous over the time because of both nature of incremental growth and diversity of technological change. Community-owned clusters are not only heterogeneous but are also non-dedicated i.e. computational resources are shared between local and parallel workload. The existing adaptive policies for dedicated cluster systems are not suitable for such conditions. This paper modifies an existing adaptive policy for dedicated heterogeneous systems so that it can be adapted to work in non-dedicated heterogeneous systems. Evaluation results show that the proposed algorithm provide substantial improvement over existing similar algorithms at moderate to high system utilizations.

Keywords:
Adaptive space-sharing scheduling, Cluster computing systems, Non-dedicated heterogeneous clusters, and Mean response time

1. Introduction

Clusters are currently a dominating supercomputing platform commonly seen in academia and industry worldwide due to their cost-effectiveness, simplicity, high-performance and scalability. Depending upon the ownership, cluster systems can be classified into two categories; dedicated cluster systems and non-dedicated cluster systems. Dedicated cluster uses a network of dedicated PCs collectively to form an effective high-performance parallel solution. On the other hand, non-dedicated cluster aims to utilize the abundant computing cycles "available" on the network of PCs to provide high-computing power. Computers in the non-dedicated clusters are owned by the user community and likely to be heterogeneous. The heterogeneity and "availability" of processing power in non-dedicated environment distinguishes itself from dedicated cluster systems. Community-owned cluster computing (CCC) systems [1][2] are example of non-dedicated heterogeneous clusters.

One of the most important challenges that must be addressed in order to realize the fullest potential of Cluster computing systems is that of designing efficient job scheduling algorithms. Space-sharing policies are commonly used to schedule parallel jobs in distributed-memory cluster systems. In space-sharing policy, parallel system of multiple processors is divided into disjoint set of processors (known as partitions) so that each partition can be assigned to a single job. In this way, number of jobs can be executed side-by-side by simultaneously providing processor partitions. The number of processors in each partition to be assigned to a job is known as partition size. The primary reason for preferring space-sharing over time-sharing for cluster systems is to avoid the cost of context switching due to frequent preemptions in time-sharing systems.

Space-sharing policies can be broadly divided into fixed, variable, adaptive and dynamic policies [3][4] based on the decision that whether the partition size once assigned to the jobs can be changed during execution time or not. In fixed policies, partition sizes are fixed by the administrator before the system actually starts operating and any modification to these partition sizes require a system reboot. Variable policies require partition sizes to be specified by the user at the time of job arrival. In adaptive policies, partition sizes are determined by the scheduler at the time of job scheduling on the basis of current system load and any available job characteristics. However partition size once assigned to a job can not be changed during job execution. In dynamic policies, partition size of a job can be changed during its execution.

Characteristics of on-line job streams that act as input workload to the job schedulers influence the performance of the schedulers. Parallel jobs can be classified into four types [3][4]; (i) Rigid, (ii) Moldable (iii) Evolving, and (iv) Malleable, depending upon the number of processor to be allocated at submission time or during execution. A rigid job demands a fixed number of processors at the time of submission and executes on these processors exclusively until completion. Moldable jobs can be made to execute on different number of processors based on the current system load. For example if system load is high, then few processors can be assigned to the moldable job and if system load then large number of processors can be allotted to the job. However this flexibility is only available at job start time and partition size cannot be reconfigured during execution. The processor requirements of both evolving and malleable jobs can be changed during execution. For evolving jobs, requirement

Manuscript received November 5, 2013
Manuscript revised November 20, 2013

changes are initiated by the application itself during the various phases of its execution. If the system cannot satisfy the job's demand, the job has to wait for exact processor allocation. For malleable jobs, the decision to change the number of processors is made by an external job scheduler.

Adaptive policies perform better than fixed-partitioning and variable-partitioning scheduling policies due to their ability adapt to the current load on the system while calculating partition-size for jobs. Adaptive space-sharing scheduling policies to schedule moldable jobs are widely studied in homogeneous parallel systems (i.e. multiprocessors and clusters) [5-12] and to less extent in heterogeneous cluster computing systems [2][13]. A common assumption in the existing adaptive policies in both the systems has been that all processors in the system are dedicated to only parallel workload. It means that processors in the system are not shared simultaneously with the local jobs executing at individual processor. In this paper we focus on proposing a scheduling algorithm to allocate processors to jobs to efficiently utilize the idle CPU cycles in a non-dedicated heterogeneous cluster computing environment.

The rest of the paper is organized as follows: Section 2 gives an overview of previous literature work related to the problem. Section 3 describes the details of the proposed solution. Section 4 describes simulation model which discusses the workload and system model used. Section 5 evaluates the performance of new policies and compares them with existing solutions and Section 6 concludes the paper.

2. Related Work

The focus of the current job scheduling research in distributed-memory multiprocessors and cluster systems is towards adaptive algorithms to schedule moldable jobs [5-12] as they have shown to achieve better mean response time than the scheduling algorithms for rigid jobs. This is due to the fact that adaptive algorithms decide the partition sizes by adapting to current system load at job scheduling time whereas rigid jobs only require a fixed number of processors resulting into increased processor fragmentation and mean response times. Dynamic policies are shown to more suitable for shared-memory parallel systems in which the associated overheads of dynamic-partitioning are outweighed by the benefits.

Adaptive scheduling algorithms for assigning partition sizes to moldable jobs have been extensively studied in homogeneous parallel systems and to less extent in heterogeneous parallel systems [2][13]. Existing adaptive algorithms in both homogeneous and heterogeneous cluster systems share one common assumption that processors are dedicated to execute only cluster applications (no other applications can be executed locally). Available adaptive policies also differ from each other by the amount of job characteristics used in making processor allocation decisions.

In [5-6] , Rosti et al. introduced several adaptive partitioning policies (known as Fixed Processors per Job (FPPJ)), Equal Partitioning with a Maximum (EPM), Insurance Policy and Adaptive Policies (known as AP1, AP2, AP3, AP4 and AP5)) for distributed-memory multiprocessors over a wide range of workload types and with different possible arrival rates. These policies try to allocate equal-sized partitions to the waiting applications since no a priori job characteristics were assumed to be available. However these policies differ from each other in how the target partition-size is computed.

Out of these adaptive policies, AP2 (known as work-conserving policy) seems to be an interesting policy that reserves one additional partition for the future job arrivals. The partition size in the AP2 policy is calculated as shown in (1).

$$Partition\ Size\ (PS) = max\left(1, ceil\left(\frac{total\_procesors}{Waiting\_jobs+1} + 0.5\right)\right) \quad (1)$$

In [7], Dandamudi and Yu show that AP2 considers only queued jobs to calculate partition size. This will lead to a situation that contravenes the principal of allocating equal-sized partitions to all jobs. Dandamudi and Yu, suggested a modified version of AP2 known as Modified adaptive policy (MAP) which considers waiting as well as running jobs to calculate partition size as shown in (2).

$$Partition\ Size\ (PS) = max\left(1, ceil\left(\frac{total\_processors}{Waiting\_jobs+(f*Running\_jobs)+1} + 0.5\right)\right) \quad (2)$$

Target partition size to be finally allocated to the waiting job is calculated using equation (3). It is the minimum of the partition size calculated using equation (2) and maximum parallelism of the job.

$$Traget\ partition\ size = min(PS, maximum\ parallelism\ of\ the\ job) \quad (3)$$

The parameter f (whose value lies between 0 and 1) is used to control the contribution of the “running” jobs to the partition size. It has been shown that the MAP policy provides significant improvement in performance over policies like AP2, ASP and ASP-max etc. that do not consider the contribution of running jobs while calculating partition size. The amount of improvement obtained is a function of parameter f, system load, and workload.

The adaptive policy proposed in [8][10] is more restrictive, in that users must specify a range of the number of processors for each job. Availability of service demand knowledge of an individual job is assumed in the paper. Schedulers will select a number which gives the best performance. Schedulers in [8][10] use a submit-time greedy strategy to schedule moldable jobs.

In [11], Srinivasan et al. have some improvement to [1][3]: (i) using schedule time-scheduler which defers the choice of partition size until the actual job schedule time instead of job submission time and, (ii) using aggressive backfilling instead of conservative backfilling.
In [12], Srinivasan et al. argue that an equal-sized partition strategy tends to benefit jobs with small computation size (light jobs). On the other hand, allocating processors to jobs proportional to the job computation size tends to benefit heavy jobs significantly. A compromise policy is that each job will have a partition size proportional to the square root of its computation size (Weight) as in (4). This equation is used to calculate partition size in an enhanced backfilling scheme proposed in [12].

$$WeightFraction_i = \frac{\sqrt{Weight_i}}{\sum_{i \in \{ParallelJobInSystem\}} \sqrt{Weight_i}} \qquad (4)$$

In [2], a variation of MAP, known as Heterogeneous Adaptive Policy (HAP) was suggested by Dandamudi and Zhou to work with heterogeneous parallel systems. The work introduced the concept of Basic Processor Unit (BPU) to differentiate the heterogeneous processors from each other. Partition sizes are allocated to the jobs on the basis of their computation power in terms of number of BPUs rather than using a physical processor level as in homogeneous systems. The research paper showed the supremacy of HAP over MAP and AP2 policies. Partition size in HAP is calculated as in equation (5) and target partition size is calculated using equation (3).

$$Partition\ Size\ (PS) = max\left(1, ceil\left(\frac{total\_BPUs}{Waiting\_jobs + (f * Running\_jobs) + 1} + 0.5\right)\right) \qquad (5)$$

In [13], Shim suggested various adaptive policies for shared heterogeneous network of workstations (NOW) considering the priority of sequential local jobs as well as the parallel jobs. No in-depth details about the working of the algorithms are provided in the paper and no comparisons are made with the existing policies. The shortcoming of this paper is that it considers only the contribution of waiting jobs to calculate the partition size which usually lead to worse results.
In [14], Doan et al. suggested priority-based adaptive policy for homogeneous PC-based cluster systems for both rigid and moldable jobs. The user can assign priority to both types of jobs. The jobs with higher priority are given preference in execution. Since rigid jobs require the fixed number of processors (e.g. partition size), so partition-function for only moldable parallel jobs is derived from equation (2) given in [7].
In [15], Abawajy proposed another adaptive policy known as SOUL for heterogeneous multi-cluster systems which calculates partition size on the basis of mean service rate of heterogeneous processors, local load at processors and maximum parallelism information of waiting jobs. It has been shown that SOUL policy tends to produce shorter mean job response times as compared to both AEP and MAP at various workloads. But no comparison between HAP and SOUL policy is available in literature.

3. Proposed Adaptive Policy

3.1 An Improved Heterogeneous Adaptive Policy (IHAP)

Using these observations and lessons, we have suggested few modifications to HAP policy which have shown good results over various policies in dedicated heterogeneous systems. The new policy is named as an Improved Heterogeneous Adaptive Policy (IHAP) to schedule jobs in non-dedicated heterogeneous cluster environment and requires only maximum parallelism (Pmax) information of jobs to calculate final target partition size for the current waiting jobs.

Partitioning-function of IHAP:
Since cluster processors can be shared between local and parallel jobs, therefore at any point of time, current available computing power for execution of parallel workload at each processor in the presence of local workload is given as in equation (6).

$$Computing\ power\ (CP_k) = BPU_k * (1 - Local\_load_k) \qquad (6)$$

In a cluster system with P processors, BPUk represents the computing power of kth processor and Local_loadk denotes the load at individual processor due to the execution of local jobs.
Ideal partition size in IHAP is then calculated on the basis of current available computing as shown in (7).

$$Partition\ Size\ (PS) = max\left(1, ceil\left(\frac{\sum_{k=1}^{P} BPU_k * (1 - Local\_load_k)}{Waiting\_jobs + (f * Running\_jobs) + 1} + 0.5\right)\right) \qquad (7)$$

It should be noted that job scheduler is invoked only at arrival and departure time of jobs. Information about local load and computing power of each processor is also collected by the job scheduler at these times. The number of BPUs finally allocated is calculated as follows in (8).

$$Traget\ partition\ size = min(PS, P_{max}) \qquad (8)$$

Job-selection rule of IHAP:
It should be noted that jobs are selected for processor-allocation from the waiting queue using Fit-Processors-First-Served (FPFS) as opposed to FCFS used in many adaptive policies [2][7][13]. Partition-size for the waiting jobs is calculated using equation (7) and (8). If the idle BPUs are less than the target partition-size for the current job, then next job from the waiting queue is searched who's target partition-size fits into the idle BPUs.

4. SIMULATION MODEL

We have implemented a discrete event simulator in VB.Net language to evaluate the performance of proposed adaptive scheduling algorithms under various workload conditions. Simulation modeling is preferred over the actual experimentation as it gave us the greater flexibility of covering a wide range of application characteristics and controlled parameters like arrival rates, system utilization etc. and allowed us to abstract away trivial details of the environment under study, which otherwise would complicate the performance evaluation procedure.

The developed simulator takes the on-line job stream as input parallel workload, executes parallel workload with the specified adaptive policy and generates the output in the form of mean response time. Response time of a job is defined as the sum of its execution time and waiting time. Waiting time of job is the difference between job arrival time and job scheduling time. Execution time is the actual time spent to execute the job.

4.1 System Model

We have used an open system model of community-owned cluster of 64 independent commodity single-processor personal computers and each computer is used in a shared mode i.e. it is able to service local sequential tasks as well as the tasks of parallel job submitted by the central job scheduler. The computers differ from each other in terms of heterogeneity in processor speeds i.e. computing power they possess. Computer and processor terms are used interchangeably in context of this paper. We assume that computers in the cluster are connected using 100Mbps Ethernet switch. Relative computing power of different physical processors is represented in terms of Basis Processing Unit (BPU) [2] which can either be derived with the help of SPECfp2000 ratings based on the processor speeds or by executing independent benchmarking programs on heterogeneous processors. We have used two types of processors in the computers of cluster system; First 32 computers contain Type I processors; Next 32 computers contain Type II processors that are twice faster than Type I processors. Hence each processor in Type I has 1 BPU and Type II processor has 2 BPUs.

4.2 Parallel Workload Model

Parallel workload model containing online stream of parallel jobs for scheduling contains three components; 1) job arrival process 2) Maximum parallelism and 3) job service demand. The job arrival process is characterized by job arrival rate (λ) and coefficient of variation of inter-arrival times (CVa). High arrival rate represents that inter-arrival time between successive jobs is small. We have modeled the job arrival process using exponential distribution with CVa equal to one.

Maximum parallelism of jobs (Pmax) indicates the maximum number of processors that can be effectively utilized by the parallel jobs. Pmax is varied from 1 to 32 using uniform distribution. Mean service demand (D) parameter is the uncorrelated cumulative mean service demand which represents the total time required to execute the job in a dedicated environment, independent of how many processors are used. Service demand of jobs is generated using 2-stage hyper-exponential distribution with coefficient of variation of service demand (CVs) greater than one. Since moldable jobs can be made to run on the varying number of processors, therefore time (tj) taken by the parallel job varies based on the number of processors (pj) assigned to it when the job starts executing. It should be noted that dj= (tj)*(pj) as we have ignored the communication and synchronization overheads, when overall mean service demand of a parallel job (dj) is distributed equally among tasks (which are always equal to “pj” processors assigned to the job) of the job.

4.3 Background Workload Model

We assume abstract model for representing load due to background jobs at each processor by hiding the internal details of arrival and execution times of sequential local jobs. Each cluster processor is assumed to service a stream of background jobs that arrive at individual computers independently. Local_load at each processor indicates the load due to the execution of sequential local jobs. As the local load increases, computing power available to service parallel workload decreases. We model the local load using discrete uniform distribution U [0%, 30%] and we assume that information about local load is only available to job scheduler at job arrival and departure times.

5. Performance Evaluation and Results

Table 1: Default parameters and values used in experiments

Parameters of Parallel Jobs	Values
Mean service demand (D)	16
Coefficient of variation (CVa) of Job arrival	1
Coefficient of variation (CVs) of Service demand	4
Number of processors in the cluster	64
Pmax	32

In this section we will evaluate the performance of proposed algorithms in terms of mean response time and also compare the simulation results with the existing approaches. In all the simulation experiments performed in this paper, 31 batches of 7000 jobs per each batch were used and results of first batch were discarded to ignore start-up effects. The number of batches is such that the

mean response times obtained have relative errors not exceeding 5% under the 90% confidence interval. The default parameters and values used in simulation experiments are for various simulation parameters shown in table 1.
Average load or utilization of the cluster system due to parallel jobs is derived using equation (7) as follows:

$$Average\ utilization = \frac{Job\ arrival\ rate * Mean\ service\ demand}{Number\ of\ processors} \quad (9)$$

5.1 Relative performance of the scheduling policies

In this section we compare the performance of the proposed adaptive scheduling policy i.e. IHAP with the HAP and MAP policy. The default value of 'f' in the partitioning-function for IHAP, HAP and MAP policies is set to 0.5 which is suggested as a reasonable value in existing similar research works [2][7].
IHAP policy tends to produce shorter MRT values at system loads of interest (i.e. at medium to high loads) as shown in figure 1. This is due to two reasons; 1) IHAP policy produce smaller partition sizes as compared to both HAP and MAP as it considers the background workload into account. 2) FPFS job-selection policy reduces processor fragmentation which exists in HAP and MAP policies due to use of FCFS as a job-selection policy.

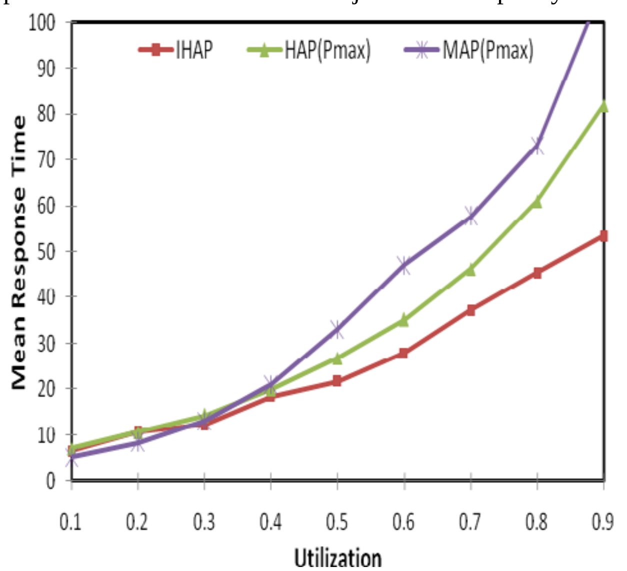

Figure 1: Performance of the scheduling policies

On the other hand, both HAP and MAP try to allocate larger partition sizes since they are not aware of any background workload. But in reality the total available computing power of all processors is much less than that of assumed by MAP and HAP. Therefore jobs have to wait for a long time to receive calculated partition sizes. HAP and MAP policies also tend to produce bigger partition sizes at low to medium system utilization since they impose no upper limit on the number of processors to be allocated to jobs. This will apparently result into allocation of large partition sizes to even smaller jobs.

5.2 Sensitivity Analysis

In this section, we study the sensitivity of the three policies to variances in inter-¬arrival and service times. When the arrival CV is varied, the service CV is held at 4. Similarly arrival CV is fixed at 1 when the performance sensitivity to service time CV is studied. The system utilization for parallel load is fixed at 80%.

5.2.1 Sensitivity to Arrival Time Variations.
The performance sensitivity of the three policies to inter-arrival CV is shown in figure 2. The mean response time increases with increasing inter-arrival CV for the three policies. The IHAP policy maintains its performance superiority over HAP and MAP policy at 80% system utilization.

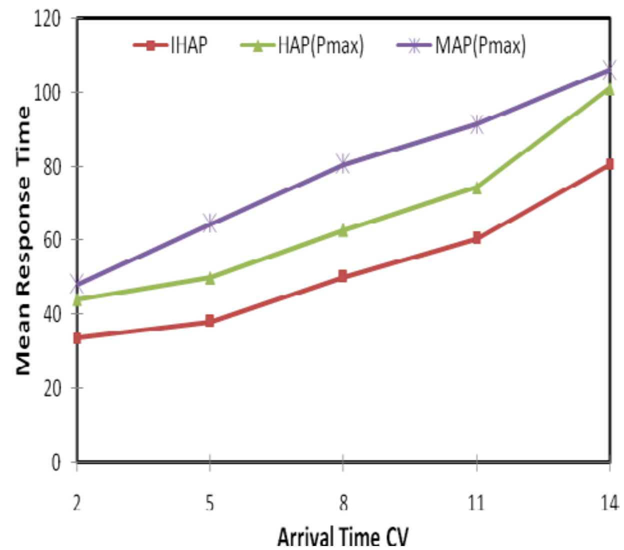

Figure 2: Sensitivity of the policies to arrival time variance

The increase in arrival time variance means the clustered arrival of jobs into the system. This also led to longer gaps in the job arrivals. The impact of variance in arrival time is more on HAP and MAP policies as shown in figure. These two policies suffer from processor fragmentation induced by the background workload and the way the partition-size is computed for the jobs. Since the partition sizes are computed on the basis of total number of BPUs (in case of HAP) and total number of processors (in case of MAP), the actual number of available BPUs (in case of HAP) and available processors (in case of MAP) can be lower than the partition-size computed. This is due to the fact that there is possibility of background tasks running on some of processors at the time and both HAP and MAP doe not consider background workload when computing partition size. But IHAP policy tend to produce smaller partition sizes due to consideration of background workload, therefore the impact of arrival time variance is reduced as compared to other two policies.

5.2.2 Sensitivity to Service Demand Variations

The figure 3 shows that MRT of the three policies increases with the increase in the variance in the service demand. With the increase in service demand variance, there will few large service demand jobs and large number of small service demand jobs. As the service time CV increases, the service demand of the larger jobs will increase even though their number goes down as a fraction of the total jobs. The impact of service time variance on HAP and MAP policies is more than the impact on IHAP policy. This is due to the fact that both HAP and MAP use FCFS as a job selection policy which is known to be sensitive of variance in service demand, to allocate processors to jobs. FCFS allocation of processors to jobs results in a situation where small jobs could be blocked by an earlier arrived large job. This problem gets more serious as the variance in service demand increases.

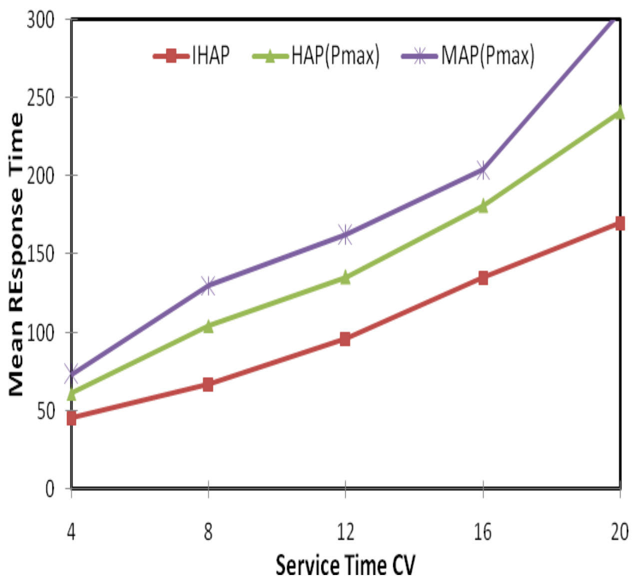

Figure 3: Sensitivity of the policies to service time variance

6. Conclusion

Space-sharing algorithms are preferred in distributed-memory cluster systems to avoid the overhead due to frequent preemptions involved in time-sharing systems. Adaptive space-sharing algorithms are used in cluster computing systems and dynamic space-sharing algorithms are more suited to shared-memory multiprocessors. Most of popular adaptive algorithms are only designed for dedicated homogeneous as well as dedicated heterogeneous cluster systems. Moreover existing adaptive policies use FCFS as a job-selection policy which is known to be sensitive to service demand variance. Hence these algorithms produce increased mean response times for workloads having high service demand variance. This paper proposes adaptive policy for non-dedicated heterogeneous cluster systems. Comparative results have shown the dominance of the proposed policy over the existing similar policies at medium to high system loads of interest. Also the policy has shown to be relatively less sensitive to service demand variance as compared to existing policies.

REFERENCES

[1] J.H. Abawajy. Parallel Job Scheduling Policies on Cluster Computing Systems. Ph.D. Thesis. Ottawa-Carleton Institute for Computer Science, Carleton University, Ottawa, Canada, November, 2003.

[2] S.P. Dandamudi and Z. Zhou, "Performance of Adaptive Space-Sharing Policies in Dedicated Heterogeneous Cluster Systems", Future Generation Computer Systems, 20(5), 895-906 (2004).

[3] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, P. Wong, Theory and practice in parallel job scheduling, in: Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, vol. 1291, Springer-Verlag, Berlin, 1997, pp. 1–34.

[4] D. G. Feitelson and L. Rudolph. Parallel Job Scheduling - A Status Report. Lecture Notes in Computer Science, Springer, Vol. 3277 (2005).

[5] E. Rosti, E. Smirni, L. W. Dowdy, G. Serazzi, and B. M. Carlson. Robust Partitioning Policies for Multiprocessor Systems. Performance Evaluation, Vol.19, 141-265 (1994).

[6] E. Rosti, E. Smirni, L.W. Dowdy, G. Serrazi, K.C. Sevcik, Processor saving scheduling policies for multiprocessor systems, IEEE Transactions on Computers 47 (2) (1998).

[7] S.P. Dandamudi and H. Yu, "Performance of Adaptive Space Sharing Processor Allocation Policies for Distributed-Memory Multicomputers", Journal of Parallel and Distributed Computing, vol. 58, pp. 109-125 (1999).

[8] W. Cirne and F. Berman. Adaptive Selection of Partition Size for Supercomputer Requests. Lecture Notes in Computer Science, Springer, Vol. 1911, 187-208 (2000).

[9] W. Cirne and F. Berman. Using Moldability to Improve the Performance of Supercomputer Jobs. Journal of Parallel and Distributed Computing, Vol. 62, 1571-1601 (2002).

[10] W. Cirne and F. Berman. A Comprehensive Model of the Supercomputer Workload. Proc. of IEEE 4th Annual Workshop on Job Scheduling Strategies for Parallel Processing (2005).

[11] S. Srinivasan, V. Subramani, R. Kettimuthu, P. Holenarsipur, and P. Sadayappan. Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs. Lecture Notes In Computer Science, Springer, Vol. 2552, 174- 183 (2002).

[12] S. Srinivasan, S. Krishnamoorthy, and P. Sadayappan. A Robust Scheduling Strategy for Moldable Scheduling of

Parallel Jobs. Proc. of 2003 IEEE International Conference On Cluster Computing (2003).
[13] Young-Chul Shim, “Performance evaluation of scheduling schemes for NOW with heterogeneous computing power”, Future Generation Computer Systems. 20(2): 229-236 (2004).
[14] V.H. Doan. An Adaptive Space-Sharing Scheduling Algorithm for PC-Based Clusters, Modeling, Simulation and Optimization of Complex Processes, pp 225-234, 2008.
[15] J.H. Abawajy, “An Efficient Adaptive Policy for High-Performance Computing”, Future Generation Computer Systems, Vol. 25, 364-370, (2009).