This paper appeared in the *Proceedings of the 8th IEEE International Conference on Cluster Computing* (Cluster 2006), Sept. 2006.

Stochastic Scheduling with Availability Constraints in Heterogeneous Clusters

Tao Xie
Department of Computer Science
San Diego State University
San Diego, California 92182
xie@cs.sdsu.edu

Xiao Qin
Department of Computer Science
New Mexico Institute of Mining and Technology
Socorro, New Mexico 87801
xqin@cs.nmt.edu

Abstract

High availability plays an important role in heterogeneous clusters, where processors operate at different speeds and are not continuously available for processing. Existing scheduling algorithms designed for heterogeneous clusters do not factor in availability. We address in this paper the stochastic scheduling problem for heterogeneous clusters with availability constraints. Each node in a heterogeneous cluster is modeled by its speed and availability, and different classes of tasks submitted to the cluster are characterized by their execution times and availability requirements. To incorporate availability and heterogeneity into stochastic scheduling, we introduce metrics to quantify availability and heterogeneity in the context of multiclass tasks. A stochastic scheduling algorithm SSAC (Stochastic Scheduling with Availability Constraints) is then proposed to improve availability of heterogeneous clusters while reducing average response time of tasks. Experimental results show that our algorithm achieves a good trade-off between availability and responsiveness.

1. Introduction

Stochastic scheduling is to investigate the problem of scheduling a set of tasks with random features. Common random features such as task processing times are usually modelled by specifying their probability distribution. Although a task's processing time is not known until it is complete, the probability distribution of task processing times are assumed to be known by the system as a priori. Stochastic scheduling could be preemptive or non-preemptive, conduct on one or on multiple processors, and be concerned with various optimization criteria.

A heterogeneous cluster consists of an array of diverse computers, called computing nodes, which are connected by a high-performance network. To date heterogeneous clusters have been emerging as popular computing platforms for computationally intensive applications with diverse computing needs. Scheduling algorithms play a key role in obtaining high performance in parallel systems like heterogeneous clusters [12]. The objective of scheduling algorithms is to map tasks onto nodes and order their execution in a way to optimize overall performance.

In scheduling theory the basic assumption is that all machines are always available for processing [17]. This assumption might be justified in some cases but it is not valid in scenarios where certain maintenance requirements, breakdowns or other constraints, which make the machines not to be available for processing, have to be considered [17]. Examples of such constraints can be found in many areas. For instance, computational nodes in heterogeneous clusters need to be maintained periodically to prevent malfunctions [10]. In this study availability is defined as the ratio of the total time a computing node is functional during a given interval to the length of the interval. Thus, the availability of a heterogeneous cluster will be degraded if one or multiple nodes are out of duty due to random breakdown or preventive maintenance. On the other hand, however, nowadays many high-performance clusters need a high availability [1][20]. For instance, military applications, 24×7 healthcare applications, and international business applications all demand an extremely high availability as severe damages or fatal errors could occur when even only one computing node becomes unavailable [1]. As such, a scheduling strategy for heterogeneous clusters has to factor in availability to deal with maintenance activities and unexpected failures. Unfortunately, conventional stochastic scheduling algorithms for heterogeneous clusters only concentrated on high throughput with the goal of reducing tasks' average response times and normally ignored the availability requirements of the tasks. It is challenging, however, to achieve high throughput and high availability simultaneously because they are two conflict objectives [1]. For example, it is unacceptable to assign a critical-task with high availability requirement to a node that provides with a high speed but low availability level. A feasible scheme is to achieve a good trade-off such that tasks' availability requirements can be met while average response time is confined to an ideal range.

In this paper we address the problem of scheduling different classes of tasks with availability constraints in heterogeneous system. We aim at developing a stochastic scheduling strategy to improve availability of heterogeneous systems while reducing average response time of multi-class tasks. The scheduling algorithm proposed in this paper can be applied to heterogeneous systems where capacity and availability constraints are known a priori.

The main contributions of this paper are: (1) an availability-latency driven stochastic scheduling scheme SSAC (Stochastic Scheduling with Availability Constraints) for multiple classes of tasks on heterogeneous clusters; (2) a system model for quantitatively measuring availability of computing nodes; (3) two types of heterogeneities: computational heterogeneity and availability heterogeneity; (4) a simulated heterogeneous cluster where the SSAC strategy is implemented and evaluated. The rest of the paper is organized as follows. In the next section we briefly introduce related works. Section 3 describes the system model, the task model and heterogeneity model. In Section 4, we propose the SSAC scheme for multiple classes of tasks running on heterogeneous clusters. We present in Section 5 experimental results based on synthetic benchmarks. Section 6 concludes the paper with summary and future directions.

2. Related work

In stochastic scheduling, a typical scenario is that m tasks have processing times that are exponentially distributed with different means and are to be processed by n identical computing nodes operating in parallel. In this case minimizing the expected makespan (the time at which all tasks are complete) is an objective function for a scheduling strategy dedicated to this scenario. Extensive research has been done in stochastic scheduling. Schopf and Berman defined a stochastic scheduling policy based on time-balancing for data parallel applications whose execution behavior can be represented as a normal distribution [19]. Cai et al. studied the problem of finding a dynamically optimal policy to process n jobs on a single machine subject to stochastic breakdowns. The problem of scheduling customers in a multi-class G/G/1 queue was addressed by Nain and Towsley [13] so as to minimize a weighted sum of the work-loads of the different classes. However, little attention has been paid to scheduling tasks with stochastic features and high system availability requirements. Recently, Chakraverty proposed a co-synthesis mechanism for generating gracefully degrading multiprocessor architectures which fulfill the dual objectives of achieving real-time performance as well as ensuring high levels of system availability [6].

Over the last decade, heterogeneous clusters have become widely used for scientific and commercial applications [8]. In recent years, the issue of scheduling on heterogeneous clusters has been addressed and reported in the literature [7][24]. Dogan and F. Özgüner developed reliable matching and scheduling algorithms for tasks with precedence constraints in heterogeneous distributed clusters [7]. Srinivasan and Jha incorporated reliability cost, defined to be the product of processor failure rate and task execution time, into scheduling algorithms for tasks with precedence constraints [23]. Ranaweera and Agrawal proposed a scalable scheduling scheme called STDP for heterogeneous systems [16]. Scheduling algorithms are of critical importance in obtaining high performance in various computing platforms [15][25]. The problem of scheduling multiple classes of tasks was motivated by a wide range of distributed applications like scalable web server systems [9]. Sethuraman et al. proposed an optimal stochastic scheduling strategy that can minimize a function of the per-class response time variances [20]. The scheduling algorithm proposed in this paper differs from theirs in that our algorithm incorporates availability constraints into scheduling. In our previous work, we studied security-aware scheduling for clusters [25] and Grids [26]. These scheduling algorithms only support homogeneous systems, limiting their applicability to heterogeneous systems. Further, these algorithms are not suitable for multi-class tasks with availability requirements. In contrast, our algorithm makes a trade-off between availability and responsiveness.

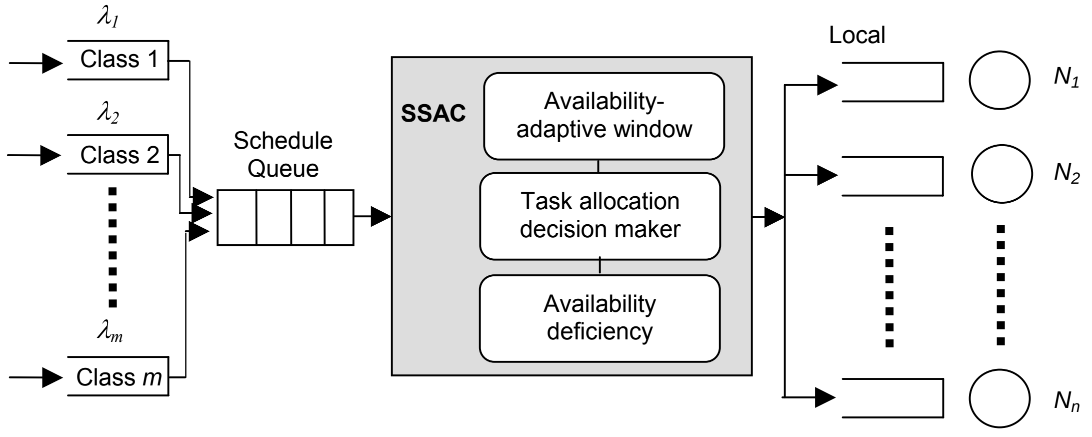

Figure 1. System model of the SSAC strategy

Some researches about improving availability of clusters were reported in the literature recently. Solter and Tripathi presented a protocol and architecture on the Sun/spl trade/ Cluster system for delivering cluster events to a high-availability cluster [21]. Leangsuksun et al. proposed concepts of integrating high availability cluster mechanism with a secure cluster infrastructure [11]. Apon and Wilbur designed an Advanced Multi Processor Network with a high availability in mind [1]. Our approach of improving availability of a heterogeneous cluster is different from the methods above. We integrated tasks' availability requirements into stochastic scheduling to achieve a good balancing between system availability and throughput measured as average response time.

3. Mathematical models

3.1. System model

In this study, we consider a queuing architecture of an n-node heterogeneous cluster in which n heterogeneous nodes are connected via a network to process independent m class of tasks submitted by m users. Let $N = \{N_1, N_2, ..., N_n\}$ denote the set of heterogeneous nodes. The system model, depicted in Figure 1, is composed of a task schedule queue, SSAC task scheduler, and n local task queues. The function of SSAC is intended to make a good task allocation decision for each arrival task to satisfy its availability requirement and maintain an ideal performance in average response time.

A schedule queue is used to accommodate incoming tasks. *SSAC* scheduler then processes all arrival tasks in a First-Come First-Served (FCFS) manner. After being handled by SSAC, the tasks are dispatched to one of the designated node $N_i \in N$ for execution. The nodes, each of which maintains a local queue, can execute tasks in parallel. The main component of the system model above is SSAC, which is composed of three modules: (1) Availability deficiency calculator; (2) Availability-adaptive window controller; and (3) Task allocation decision maker. The availability deficiency calculator is used to calculate discrepancies between an arrival task's availability requirement and the availability value that each node offers. The function of availability-adaptive window controller is to vary size of the window to discover a suitable node for the current arrived task so that (1) its availability demands can be well met; (2) the execution time can be as small as possible. To illustrate how availability-adaptive window controller works, we give an example as below.

In Figure 2 we assume that there are 8 nodes in the cluster. The first row shows the execution time for an arrival task on the 8 nodes in seconds. Note that the execution of each node for a particular task is an expected execution time. The second row displays the availability levels that the 8 nodes can offer. The third row is a node list sorted by the task's execution time in a non-decrease order. The size of availability-adaptive window is 4, which means SSAC will select a node that can deliver the best availability within the first four candidate nodes. If availability-adaptive window controller cannot find an idea node in terms of availability, it will automatically enlarge the window to expand the search range. However, large window size will result in a long execution time for the task in a high probability because the execution time increases when the size of the window enlarges.

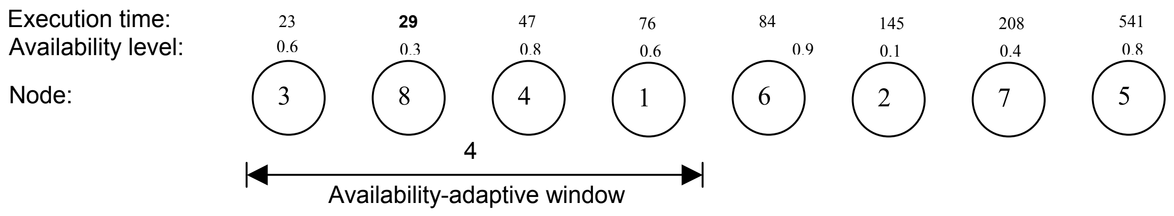

Figure 2. Example sorted node list with availability-adaptive window 4

After retrieving information like degree of availability deficiency on each node and the size of availability-adaptive window for the current task from the corresponding modules, the task allocation decision maker will decide which node will be assigned to the task. Each node in the system model above is inherently heterogeneous in both computation and availability. Computational heterogeneity means that for each task the execution time on different nodes is distinctive. While each task has an availability request, each node offers the availability with different levels. The level of availability provided by a node is normalized in the range from 0 to 1.0.

3.2. Tasks with availability requirements

For future reference, we summarize the notations for availability in Table 1.

We consider a heterogeneous cluster system where arrival tasks are independent of one another. There are m classes of tasks submitted to the system by m users. Each class of tasks requires a common availability level specified by a user. Values of availability levels are normalized in the range from 0 to 1.0. For example, a critical task may specify availability level 1.0 in its request, meaning that this task should be assigned to a node that can provide 100 percent availability. It will take a risk for being disturbed during its execution, otherwise. Suppose there are m different users and each of them keeps submitting a class of tasks. The arrival pattern of tasks in class i follows a Poisson process with rate λ_i (see Figure 1).

Suppose there is a task T_i submitted by a user, T_i is modeled as a set of rational parameters, e.g., $T_i = (a_i, E_i, f_i, av_i)$, where a_i and f_i are the arrival and expected finish times, and E_i is a vector of expected execution times for task T_i on each node in N, and $E_i = (e_i^1, e_i^2, \ldots, e_i^n)$. Suppose T_i's requirement of availability is av_i.

Since arrival patterns and service rates can be estimated by code profiling and statistical prediction [3], it is assumed in this study that the arrival patterns and service rate is known a priori.

Table 1. Notation of system model

Notation	Explanation
m	Number of classes of tasks
n	Number of nodes in the system
i	One particular class of tasks, $1 \le i \le$
j	One particular node, $1 \le j \le n$
λ_i	Arrival rate of the ith class of tasks
λ	Task arrival rate of the entire system
p_{ij}	Probability that a task of the ith class is dispatched to node j
μ_{ij}	Service rate of tasks of the ith class on node j
ρ_i	Service utilization of all tasks of class
ϕ_j	Service utilization of node j
ϕ	Total service utilization of the system
ξ_j	Probability that node j is available
a_i	Availability requirement of tasks of class i, $0 \le a_i \le 1$
δ_j	Availability deficiency of node j
Λ_j	Task arrival rate of the jth node
θ_j	Unavailable rate of node j
α	A parameter to control the value of
A	System availability
d_{ij}	Discrepancy between ξ_j and a_i
ES_j	Mean service time of node j
ES_j^2	Mean-square service time of node j
s_{ij}^2	Mean-square service time of class i on node j
TC_i	Expected response time of tasks in class i

T	Expected response time of all classes

Without loss of generality, we assume that tasks of the *i*th class arrive according to a Poisson process with rate λ_i. All classes of tasks arrive at the system at total rate of $\lambda = \sum_{i=1}^{m} \lambda_i$. Let p_{ij} be the probability that a task of the *i*th class is dispatched to node *j*, where $1 \le j \le n$. Thus, the task arrival rate of the *j*th node is computed by $\sum_{i=1}^{m} p_{ij}\lambda_i$. The service rate of tasks of class *i* on node *j* is denoted by μ_{ij}, and the corresponding mean service time is given by $1/\mu_{ij}$. The service utilization of class *I* is expressed as $\rho_i = \sum_{j=1}^{n} [p_{ij}(p_{ij}\lambda_i/\mu_{ij})] = \sum_{j=1}^{n} (p_{ij}{}^2\lambda_i/\mu_{ij})$. The service utilization at node *j* is given by $\phi_j = \sum_{i=1}^{m} (p_{ij}\lambda_i/\mu_{ij})$. The total service utilization of the system can be calculated by $\phi = \sum_{j=1}^{n} \phi_j$.

Let ξ_j denote the probability that node *j* is available for computation. We denote a_i as availability requirement of class *i*. To quantify availability of a heterogeneous system, we introduce the concept of availability deficiency. Thus, the availability deficiency of node *j* is expressed by

$$\delta_j = \sum_{i=1}^{m} \frac{p_{ij}\lambda_i}{\Lambda_j} d_{ij} = \frac{1}{\Lambda_j} \sum_{i=1}^{m} p_{ij}\lambda_i d_{ij} \quad , \qquad (1)$$

where $\Lambda_j = \sum_{i=1}^{m} p_{ij}\lambda_i$, $d_{ij} = \begin{cases} 0, \text{if } \mathrm{a}_i \le \xi_j \\ \mathrm{a}_i - \xi_j, \text{otherwise} \end{cases}$

Equation 1 measures the discrepancy between the availability of node *j* and the availability requirements of tasks allocated to node *j*. Although the availability deficiency reflects a satisfaction degree in availability, the availability deficiency is inadequate to evaluate system availability for all the classes of tasks during their executions. The availability of the system for class *i* is calculated by $\exp\left[-\sum_{j=1}^{n} p_{ij}\frac{\theta_j}{\mu_{ij}}\right]$, where θ_j is the unavailable rate of node *j*. The unavailable rate is expressed as: $\theta_j = 1 - \exp(-\alpha(1-\xi_j))$. Note that the rate model is just for illustration purpose only, and it can be replaced by any unavailable rate model or using any reasonable parameter α. The system availability can be obtained below

$$A = \sum_{i=1}^{m} \left\{ \frac{\lambda_i}{\lambda} \exp\left[-\sum_{j=1}^{n} \left(p_{ij} \frac{\theta_j}{\mu_{ij}} \right) \right] \right\}. \qquad (2)$$

We model each node in the system as a single M/G/1 queue. Hence, the average response time of node *j* can be calculated as

$$TN_j = ES_j + \frac{\Lambda_j ES_j^2}{2(1-\phi_j)}, \qquad (3)$$

where ES_j and ES_j^2 are the mean and mean-square service time of node *j*, respectively. They can be calculated by the following equation:

$$ES_j = \sum_{i=1}^{m} \left(\frac{p_{ij}\lambda_i}{\Lambda_j} \cdot \frac{1}{\mu_{ij}} \right) = \frac{1}{\Lambda_j} \sum_{i=1}^{m} \left(\frac{p_{ij}\lambda_i}{\mu_{ij}} \right) \quad ,$$

$$ES_j^2 = \sum_{i=1}^{m} \left(\frac{p_{ij}\lambda_i}{\Lambda_j} \cdot s_{ij}^2 \right) = \frac{1}{\Lambda_j} \sum_{i=1}^{m} \left(p_{ij}\lambda_i s_{ij}^2 \right) \qquad (4)$$

Where s_{ij}^2 is the mean-square service time of class *i* on node *j*.

Based on Equation 3, the expected response time of all the classes is given as follows

$$T = \sum_{i=1}^{m} \left(\frac{\lambda_i}{\lambda} TC_i \right), \qquad (5)$$

where TC_i is the expected response time of class *i* tasks.

Now we formulate the stochastic scheduling problem as a trade-off problem between availability and mean response time. Thus, the objective of the proposed scheduling algorithm is to maximize system availability (see Equation 2) and to minimize mean response time of submitted tasks (see Equation 5).

3.3. Heterogeneity model

We describe two types of heterogeneities in this subsection. The computational weight of class *i* on node *j* is defined as a ratio between its service rate on node *j* and the fastest service rate in the system. That is, the computational weight is expressed by $w_{ij} = \mu_{ij} \Big/ \max_{k=1}^{n} (\mu_{ik})$. The computational heterogeneity of the *i*th class, i.e. HC_i, can be measured by the standard deviation of the computational weights. Thus, we have

$$HC_i = \sqrt{\frac{1}{n}\sum_{j=1}^{n} \left(\overline{w_i} - w_{ij}\right)^2}, \overline{w_i} = \left(\sum_{j=1}^{n} w_{ij} \right) \Big/ n \qquad (6)$$

where $\overline{w_i}$ is average computational weight.

The computational heterogeneity can be expressed by $HC = \frac{1}{m}\sum_{i=1}^{m} HC_i$.

The heterogeneity of availability in a heterogeneous system is written as

$$HA = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\overline{\xi} - \xi_j\right)^2}, \overline{\xi} = \left(\sum_{j=1}^{n}\xi_j\right)\Big/n . \quad (7)$$

4. The SSAC algorithm

We now propose the scheduling algorithm, which is intended to determine probability $\{p_{ij}\}_{1\le i\le m,\ 1\le j\le n}$ in a way to improve the system availability and reduce mean response time. Our algorithm relies on the following proposition, which can be easily proved based on proposition 2.1 in [20].

Proposition: Given an *m*-class *M/G/1* queue and an *n*-node heterogeneous system, class *i* has arrival rate λ_i and service rate μ_{ij} on node *j*. The scheduling policy on node *j* that gives a higher priority to class *i* over *k* whenever $\mu_{ij} \ge \mu_{kj}$ minimizes the expected response time $T = \sum_{i=1}^{m}\left(\frac{\lambda_i}{\lambda}TC_i\right)$ (see Equation 5).

```
1. Sort and label classes such that
   $\lambda_1\Big/\sum_{j=1}^{n}\mu_{1j} \ge \lambda_2\Big/\sum_{j=1}^{n}\mu_{2j} \cdots \ge \lambda_m\Big/\sum_{j=1}^{n}\mu_{mj}$;
2. for each class i do
3.    Initialize availability deficiency and response
      time for class i, i.e., $d \leftarrow \infty$, $TC \leftarrow \infty$;
4.    create a set of nodes $N_i$, where
      node $j \in N_i$  if $a_i \le \xi_j$;
5.    for each node j in $N_i$ do
6.        calculate availability deficiency of
              class i on node j, $d_{ij}$;
7.        calculate expected response time of
              class i, $TC_i$; (see Equation 8)
8.        if $d_{ij} < d$ or ($d_{ij} = d$ and $TC_j < TC$) then
9.            $d \leftarrow d_{ij}$ ; $TC \leftarrow TC_j$; $\pi \leftarrow j$;
10.       end if
11.   end for
12.   if the system is balanced then
13.       $p_{i\pi} \leftarrow 1$;
14.       allocate class i to node $\pi$;
15.   else
16.       Allocate class i to the lightly loaded
          node;
17.   end if
18.end for
```

Figure 3.The SSAC algorithm

It is assumed that the classes are labelled such that $\lambda_1\Big/\sum_{j=1}^{n}\mu_{1j} \ge \lambda_2\Big/\sum_{j=1}^{n}\mu_{2j} \cdots \ge \lambda_m\Big/\sum_{j=1}^{n}\mu_{mj}$. This assumption is valid because the task classes can be sorted and relabelled in the first step of the algorithm. For tasks of class *i*, the expected response time on node *j* can be approximated by

$$TC_i = W_i + \frac{1}{\mu_{ij}} = \frac{\sum_{i=1}^{m} p_{ij}\lambda_i E(s_i^2)}{2(1-\sum_{l<i}\rho_{lj})(1-\sum_{l\le i}\rho_{lj})},$$

where $\rho_{lj} = \frac{p_{lj}\lambda_j}{\mu_{lj}}$. (8)

The SSAC algorithm is depicted in Figure 3. The algorithm aims to improve availability while maintaining low average response time for multi-class tasks on heterogeneous systems. First, SSAC gives classes with higher service utilization higher priority. Thus, SSAC sorts and re-labels all the classes in a way that $\lambda_1\Big/\sum_{j=1}^{n}\mu_{1j} \ge \lambda_2\Big/\sum_{j=1}^{n}\mu_{2j} \cdots \ge \lambda_m\Big/\sum_{j=1}^{n}\mu_{mj}$ (see Step 1). Step 4 identifies all nodes that can meet the availability demand of tasks of class *i*. Steps 5-10 are at the core of the proposed algorithm. In particular, Steps 6 and 7 leverage Eqs. 1 and 9 to estimate the availability deficiency and expected response time of class *i* on node *j*. Step 8 makes an effort to improve system availability by reducing availability deficiency. In the case there is no way to further reduce the availability deficiency, Step 8 endeavors to minimize the expected response time of class *i*. When the load of the system is balanced, Steps 13 and 14 allocate tasks of class *i* to node π, which yields the highest system availability. Importantly, the average response time is further reduced by the algorithm through static load balancing (see Steps 12 and 16).

Theorem 1. The time complexity of SSAC is *O*(*nm*), where *n* is the number of nodes, *m* is the number of task classes.

Proof. The time complexity calculating availability deficiency and response time for node j is *O*(*n*) (Step 6 and 7). Sorting the classes of tasks time a non-decreasing order (Step 1) will take *O*(*mlgm*) since we only have m classes. For other steps, they only consume *O*(*1*). Thus, the time complexity of the SSAC algorithm is as follows: $O(m)(O(n))+ O(mlgm)= O(mn)+O(mlgm)= O(mn)$. □

5. Simulations

Using simulation experiments based on synthetically generated workload, we evaluate in this section the performance of the SSAC algorithm. Task arrival rate λ and task execution time range (ETR) are two important workload parameters (see Table 2). We assume that task arrival times abide by Poisson distribution and task execution times follow Uniform distribution. We evaluate SSAC along with two existing algorithms under a wide range of system workload conditions by varying λ and number of nodes. To reflect the heterogeneity of the simulated distributed system, we translated the "execution time" of each task from a single value to a vector with n (number of nodes) elements based on the heterogeneity model described in Section 3. In purpose of revealing the strength of SSAC, we compared it with two well-known scheduling algorithms, namely, *Min-Min* and *Sufferage* [22]. *Min-Min* and *Sufferage* are non-preemptive task scheduling algorithms, which schedule a stream of independent tasks onto a heterogeneous distributed computing system. They are representative dynamic scheduling algorithms for distributed systems and were successfully applied in real world distributed resources management systems such as SmartNet. The two algorithms are briefly described below.

(1) MINMIN: For each submitted task, the node that offers the earliest completion time is tagged. Among all the mapped tasks, the one that has the minimal earliest completion time is chosen and then allocate to the tagged node.

(2) SUFFERAGE: Allocating a node to a submitted task that would "suffer" most in terms of completion time if that node is not allocated to it.

Table 2. Characteristics of system parameters

Parameter	Value (Fixed) - (Varied)
Number of sites	(16) – (16, 32,64,128)
Task arrival rate λ (Poisson dist.)	(1.0) – (0.2, 0.4, 0.6, 0.8, 1.0)
Task execution time range (ETR)	(1, 500) second
Node availability level (Uniform dist.)	(0.1 – 1.0)
Task availability level (Uniform dist.)	(0.1 – 1.0)
Computational heterogeneity	1.08
Availability heterogeneity	0.22

5.1. Simulation setup

Table 2 summarizes the key configuration parameters of the simulated distributed system used in our experiments. The performance metrics we used include: *Availability* (see Equation 2), *Availability shortage* (defined as the mean discrepancy between availabilities requested by tasks and availabilities provided by nodes in the distributed system), *Node utilization* (defined as the percentage of total task running time out of total available time of a given node), *Average response time* (see Equation 5).

5.2. Overall performance comparisons

The goal of this experiment is two fold: (1) to compare the proposed SSAC algorithm against the two heuristics, and (2) to understand the sensitivity of SSAC to the task arrival rate λ.

Figure 4 shows the simulation results for the three algorithms on a distributed system with 16 nodes. We observe from Figure 4a that SSAC significantly outperforms the two heuristics in terms of Availability, whereas MINMIN and SUFFERAGE algorithms exhibit similar performance. We attribute the performance improvement of SSAC over MINMIN and SUFFERAGE to the fact that SSAC is an availability-adaptive scheduler and judiciously assigns a task to a node not only considering its computational time but also its availability demands. Similar observations can be made from Figure 4b, where SSAC has a much better performance in Availability shortage. The lower Availability shortage, the better performance achieved in availability satisfaction. As for the performance of Average response time (Figure 4c), SSAC is slightly worse than the two existing algorithms by 5.7% on average. However, the performance improvement of SSAC over MINMIN and SUFFERAGE in Availability is 73.3% on average. In addition, SSAC achieves a better performance in Node utilization as well (Figure 4d).

5.3. Scalability

This experiment is intended to investigate the scalability of the SSAC algorithm. We scale the number of nodes in a heterogeneous distributed system from 16 to 128. Figure 5 plots the performances as functions of the number of nodes in the simulated distributed system. The results show that the SSAC approach exhibits good scalability.

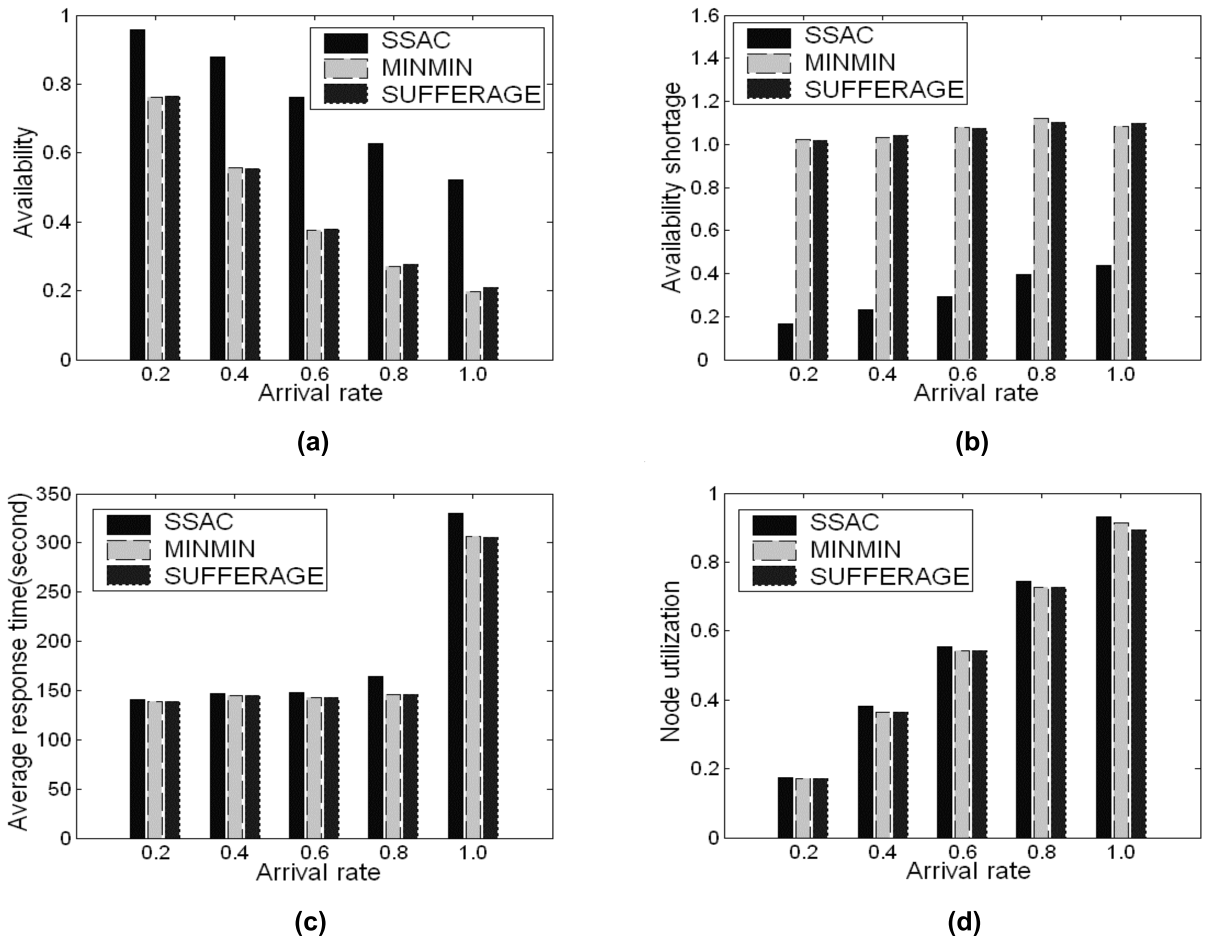

Figure 4.Performance impact of task arrival rate λ

Figures 5a and Figure 5b show the improvement of SSAC in *Availability* and *Availability shortage* over the other two heuristics. It is observed from Figure 5a that the amount of improvement becomes more prominent with the increasing value of node number. This result can be explained by the non-availability-awareness nature of MINMIN and SUFFERAGE, which merely select a node for a task without considering the task's availability demands. Conversely, SSAC can achieve a much higher performance when there are more nodes available in the system. This is because with a high probability SSAC can find a node that meets a task's availability demands well when there are more nodes to be chosen. For all the three algorithms, their performances improve in terms of *Average response time* (Figure 5c). This can be readily understood because more nodes result in a low value for *Average response time*.

6. Summary and future work

In this paper, we address the stochastic scheduling problem for heterogeneous systems with availability constraints. While different classes of tasks are characterized by their execution times and availability requirements, each node in a heterogeneous system is modeled by its speed and availability. We introduce metrics to quantify availability and heterogeneity in the context of multi-class tasks. To incorporate availability and heterogeneity into scheduling, we have proposed a stochastic scheduling. Our scheduling algorithm is geared to enhance availability of heterogeneous systems while reducing average response time of multi-class tasks. Empirical results show that the novel algorithm improves performance in availability over existing schemes for heterogeneous systems. In future research, the heuristic will be extended to schedule parallel applications. This work can be accomplished by factoring in precedence constraints among tasks and communication availability.

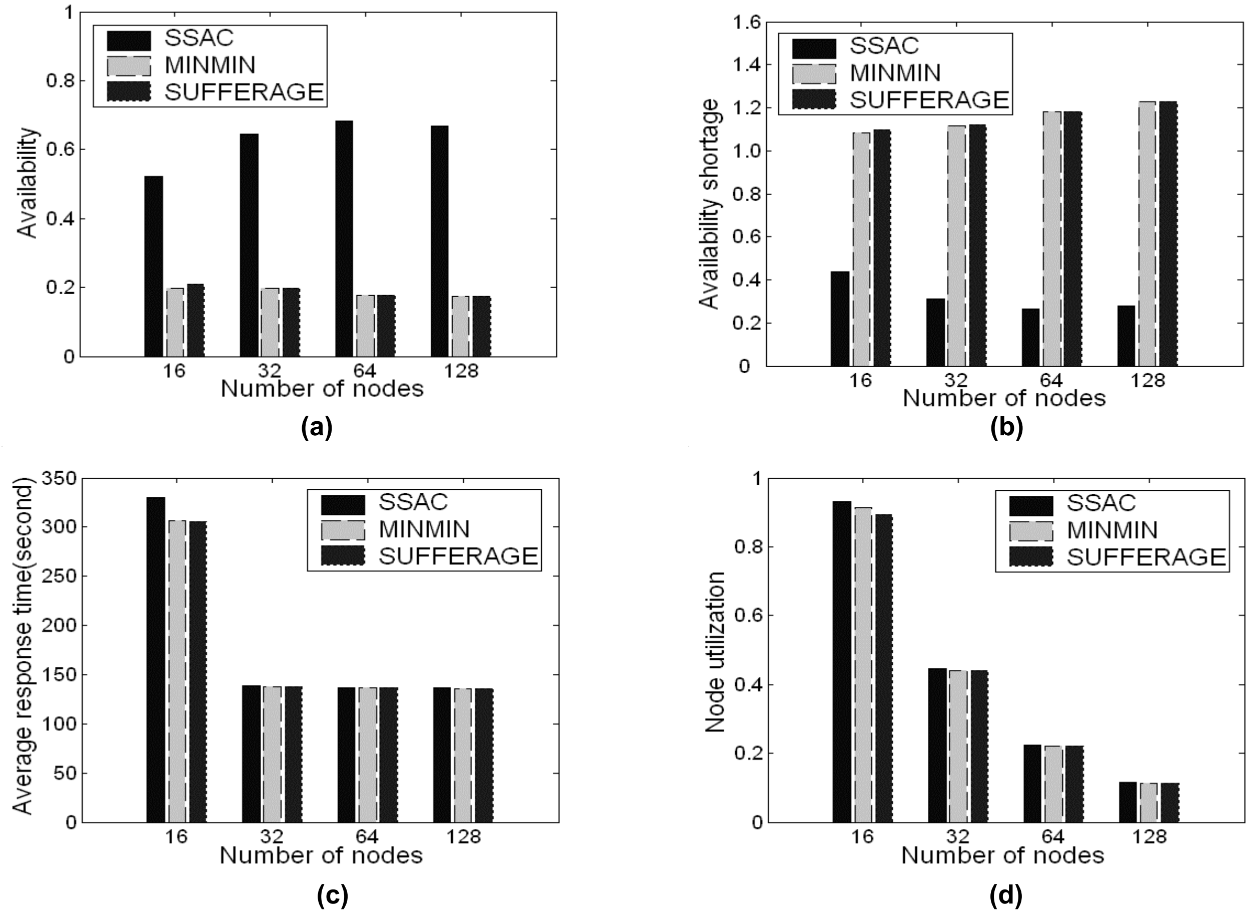

Figure 5. Performance impact of number of nodes

References

[1] A. Apon and L. Wilbur, "AmpNet - a highly available cluster interconnection network," *Proceedings IEEE Intl' Symp. Parallel and Distributed Processing*, April 22-26, 2003.

[2] A.C. Arpaci-Dusseau, "Implicit Co-scheduling: Coordinated Scheduling with Implicit Information in Distributed Systems," *ACM Trans. on Computer Systems*, Vol.19, No. 3, pp.283-331, Aug. 2001.

[3] T. D. Braun *et al.*, "A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems," *Proc. Workshop Heterogeneous Computing,* pp.15-29, Apr. 1999.

[4] X. Cai, X. Wu, and X. Zhou, "Dynamically optimal policies for stochastic scheduling subject to preemptive-repeat machine breakdowns," *IEEE Transactions on Automation Science and Engineering*, Vol. 2, Issue 2, April 2005.

[5] T.L. Casavant and J.G. Kuhl, "A Taxonomy of Scheduling in General-purpose Distributed Computing Systems," *IEEE Trans. Software Engineering*, Vol.14, No.2, pp.141-154, Feb. 1988.

[6] S. Chakraverty, "Cosynthesis on multiprocessor architectures with high availability," *Proceedings 17th International Conference on VLSI Design*, pp. 927-932, 2004.

[7] A. Dogan and F. Özgüner, "Reliable Matching and Scheduling of Precedence-Constrained Tasks in

Heterogeneous distributed computing," *Proc. Int'l Conf. Parallel Processing*, pp. 307-314, 2000.

[8] A. Dogan and F. Özgüner, "LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems," Proc. Int'l Conf. Parallel Processing, pp.352-359, B.C., Canada, 2002.

[9] G. Hunt, G. Goldszmidt, R. King, and R. Mukherjee, "Network Dispatcher: A Connection Router for Scalable Internet Services," *Proc. Int'l World Wide Web Conf.*, April 1998.

[10] H. C. Lau and C. Zhang, "Job Scheduling with Unfixed Availability Constraints," *Proc. 35th Meeting of the Decision Sciences Institute* (DSI), 4401-4406, Boston, USA, November 2004.

[11] C. Leangsuksun, A. Tikotekar, M. Pourzandi, and I.Haddad, "Feasibility study and early experimental results towards cluster survivability," *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, pp. 77-81, 2005.

[12] M. Maheswaran and H.J. Siegel, "A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems," *Proc. the Seventh Heterogeneous Computing Workshop*, pp.57-69, 1998.

[13] P. Nain and D. Towsley, "Stochastic scheduling in a multiclass G/G/1 queue," *Proceedings of the 31st IEEE Conference on Decision and Control*, pp 3340-3341, 1992.

[14] D.-T. Peng and K.G. Shin, "Optimal scheduling of cooperative tasks in a distributed system using an enumerative method," *IEEE Trans. Software Engineering*, Vol.19, No.3, pp. 253-267, March 1993.

[15] X. Qin and H. Jiang, "A Dynamic and Reliability-driven Scheduling Algorithm for Parallel Real-time Jobs on Heterogeneous Clusters," *Journal of Parallel and Distributed Computing,* Vol. 65, No. 8, pp.885-900, August 2005.

[16] S. Ranaweera, and D.P. Agrawal, "Scheduling of Periodic Time Critical Applications for Pipelined Execution on Heterogeneous Systems," *Proc. Int'l Conf. Parallel Processing*, pp. 131-138, Sept. 2001.

[17] G. Schmidt, "Scheduling with limited machine availability," *European Journal of Operational Research*, pp. 1-15, 121, 2000.

[18] Journal of Operational Research 121 (2000) 1–15.

[19] J.M Schopf and F. Berman, "Stochastic Scheduling," *Proceedings of the ACM/IEEE Conf. Supercomputing*, 13-18 Nov. 1999.

[20] J. Sethuraman and M. S. Squillante, "Optimal Stochastic Scheduling in Multicalss Parallel Queues," *Proc. ACM Sigmetric Conf.*, May 1999.

[21] N.A. Solter and A. Tripathi, "Architecture and protocol for reliable event delivery to clients of a high-availability cluster," *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, April 2004.

[22] S. Song, Y.-K. Kwok, and K. Hwang, "Trusted Job Scheduling in Open Computational Grids: Security-Driven Heuristics and A Fast Genetic Algorithms," *Proc. Int'l Symp. Parallel and Distributed Processing*, 2005.

[23] S. Srinivasn and N. K. Jha, "Safty and Reliability Driven Task Allocation in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems*, Vol.10, No.3, pp. 238-251, Mar. 1999.

[24] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Sys.*, Vol.13, No.3, Mar. 2002.

[25] T. Xie and X. Qin, "A New Allocation Scheme for Parallel Applications with Deadline and Security Constraints on Clusters," *Proc. IEEE Int'l Conf. Cluster Computing*, Boston, USA, Sept. 2005.

[26] T. Xie and X. Qin, "Enhancing Security of Real-Time Applications on Grids through Dynamic Scheduling," *Proc. 11th Workshop Job Scheduling Strategies for Parallel Processing*, MA, June 2005.