Journal of Internet Technology, vol. 15, no. 1, pp. 115-124, January 2014.

H2ACO: An Optimization Approach to Scheduling Tasks with Availability Constraint in Heterogeneous Systems

ZHAO TONG[1], *KENLI LI*[1], *ZHENG XIAO*[1], *XIAO QIN*[2]
1 *College of Information Science and Engineering*
Hunan University
CHINA
{tongzhao1985, lkl, zxiao}@hnu.edu.cn
2 *Department of Computer Science and Software Engineering*
Auburn University, USA
http://www.eng.auburn.edu/~xqin
xqin@auburn.edu

Abstract

An efficient resource management mechanism is important in a heterogeneous distributed system to discover available resources, to allocate an appropriate subset of resources to applications, and to map data or tasks onto selected resources. The key component, task scheduling, draws our attention. Makespan is the principal concern of many existing researches. But, other QoS requirements are also important in more and more realistic applications. For example Cloud Computing is expected that the service provider is reliable, robust, or highly available. In this study, we develop H2ACO (Hybrid Heuristic-Ant Colony Optimization) which makes a good trade-off between availability and makespans for heterogeneous distributed systems running multiclass applications. H2ACO comprises two key components: (1) an ant optimization algorithm which makes initial scheduling decisions; (2) an availability-aware scheduling mechanism which optimizes initial schedules offered by the first component. The experiment results indicate that compared with two existing solutions (PSO and SSAC), H2ACO significantly improves the availability and performance of multiclass tasks running in heterogeneous systems.

***Keywords*:** Heterogeneous systems; Task scheduling; QoS Scheduling; Availability; Optimization.

1 Introduction

A heterogeneous distributed system (HDS) is a group of processors connected via a high speed network, which supports the execution of parallel applications. The efficiency of executing parallel applications on distributed systems depends how the tasks are scheduled among the processors. Task scheduling is aiming at optimizing some certain performance metric. Generally, it involves two parts: allocating tasks from applications to the set of available processors and ordering the execution of the tasks on each processor. The task scheduling problem is an NP-complete problem [1, 2].

The performance of heterogeneous systems can be improved by a wide variety of scheduling algorithms [3, 4]. This paper focuses on scheduling of multiclass tasks. A multiclass application consists of tasks of multiple classes, which are characterized by their distinctive arrival rates, execution time distributions and availability requirements.

Existing scheduling algorithms [5, 6] tailored for multi-class applications running in heterogeneous systems only concentrated on performance. They ignore availability constraints of multiclass tasks. Processors are not continuously available for computation. Recent study [7] shows that it is challengeable to achieve both high performance and high availability because these two requirements are conflicting objectives. For example, it is unwise to allocate tasks with availability constraints to high-performance computing nodes which offer low availability. In this study, we aim at developing a scheduling scheme that guarantees the availability requirements of multiclass tasks while achieving high performance of heterogeneous systems.

Qin and Xie proposed a novel strategy called SSAC (Stochastic Scheduling with Availability Constraints) to schedule multiclass tasks in heterogeneous systems [7]. However, we observed that the SSAC algorithm assigns a large number of tasks to a node with a high availability level. As a result, system performance achieved by SSAC may significantly decline due to imbalanced load. To address this performance issue, we presented an approach which makes a good trade-off between availability and makespan for heterogeneous systems.

In this paper, we propose an algorithm using evolutionary computation to schedule multiclass tasks running in heterogeneous systems where the computing capacity and availability constraints are known as a priori. The four major contributions of this study are listed below:

- For the first time, scheduling optimization is formally formulated as an integer programming problem in the context of heterogeneous distributed systems with the availability constraints.
- An optimization approach is presented to increase system availability and meanwhile to reduce scheduling makespans.
- A new evolutionary-computation based scheduling algorithm is proposed to incorporate the above optimization scheme to schedule multiclass tasks with availability constraints in heterogeneous computing systems.
- It is demonstrated through performance comparisons that the proposed algorithm improves both performance and availability of heterogeneous systems supporting multiclass applications.

The rest of this paper is organized as follows. Section 2 reviews related work on scheduling techniques in heterogeneous systems. Section 3, shows a motivational example. Section 4 describes the system model and the formalization of the scheduling problem with availability constraint. Section 5, presents the idea of our optimization scheme to solve task scheduling problem in heterogeneous systems. The experiments and results are given in Section 6. Finally, conclusions are drawn in Section 7.

2 Related work

Many existing scheduling solutions require an assumption that machines have 100 percent availability. However, this assumption is not valid in real-world systems, where maintenance requirements, breakdowns, or other constraints make the machines unavailable for a certain period of time [8]. Importantly, many high-performance applications (e.g., military applications, 24*7 healthcare applications, and international business applications) require computing platforms to have extremely high availability [9, 10]. This is because catastrophic consequences may occur, even if only one computing node becomes unavailable [9]. Therefore, any practical scheduling mechanism must make an effort to improve the availability of heterogeneous computing systems.

Recently, researchers have developed task scheduling strategies for computing systems with availability constraints. For example, Wang et al. studied a single-machine scheduling problem in which tasks have availability constraints [11]. Their scheduling scheme aims to minimize total weighted completion times with several unavailability periods for preemptive jobs. Kacem and Chu proposed a single-machine scheduling algorithm to minimize total weighted completion time for jobs with a single unavailability period [12]. After discovering a new lower bound for the problem, Kacem and Chu designed two heuristics called WSPT and MWSPT (i.e., modified weighted shortest processing time), which have the same worst case error bound. Recently, Kacem and Chu improved the branch-and-bound algorithm using new heuristics offering lower bounds [13-14] proposes a new job-scheduling algorithm called first come first served plus predictor (FCFSPP). This scheduling algorithm is based on an existing resource availability prediction method that anticipates the future availability of resources to help make reliable job allocation decisions.

The existing algorithm SSAC [7] has been proved to be a good trade-off between availability and responsiveness while maintaining a good performance in the average response time of multiclass tasks. But the makespan may be influenced due to the load imbalance. [15] proposed an approach to trade off makespan and availability by using quantum-behaved particle swarm optimization. The performance improves a little bit. And this paper tries to provide much better performance.

3 Motivation example

To improve the availability of multiclass tasks running in a heterogeneous system without degrading performance, the SSAC [7] can make good task-allocation decisions for each class of tasks to satisfy their availability requirements while achieving an ideal performance in terms of the response time. This approach, of course, tends to distributed tasks over some computing nodes providing high availability to fully satisfy the tasks' availability requirements. As a result, a large number of tasks might be assigned to a single node with the highest availability. Thus, from the systematic prospect, the remained availability of the entire system and the makespan of tasks may be adversely affected due to such a load imbalance. We make use of the following motivational example to demonstrate that the above availability-aware scheduler may lead to imbalanced load and poor performance.

Considering a small-scale scheduling problem with 5 tasks and 3-node heterogeneous system, the

availability levels of these three computing nodes are 0.98, 0.86 and 0.6, respectively. Other parameters are listed in Table 1, in which the columns n_0, n_1, and n_2 represent execution time of a task on the three nodes, respectively, and the column α_i represents the availability requirement of tasks.

Table 1 Parameters and their values in example 1

5 tasks on 3 nodes	n_0	n_1	n_2	α_i
t_0	3	5	8	0.50
t_1	6	8	11	0.60
t_2	2	5	4	0.70
t_3	3	6	2	0.82
t_4	5	3	7	0.99

The scheduling result of an existing scheme (i.e., SSAC) is shown in Table 2, which indicates that the makespan (the longest completion time of all the nodes) is 13 (3+2+3+5=13). The result shows that to enhance system availability, the availability cost must be reduced. The scheduling decision illustrated in Table 2 increased the availability at the cost of performance, as load imbalance is caused by assigning a large number of tasks to the node with the highest availability. We observe that if t_0 is moved to n_2 from n_0 (see Table 3), a better scheduling decision is obtained that offers higher system availability and shorter makespan.

Table 2 Makespan of the schedule from SSAC

5 tasks on 3 nodes	n_0	n_1	n_2
t_0	3	-	-
t_1	-	8	-
t_2	2	-	-
t_3	3	-	-
t_4	5	-	-

Table 3 shows an improved schedule made by an optimization algorithm. The makespan of the improved schedule is 10, which is 30% smaller than that generated by the existing scheme. In addition, the system availability offered by the improved schedule is higher than that shown in Table 2, since the availability cost of t_0 on the new node is smaller.

Therefore, [7] proposes a simple detector to make up for load imbalance. This detector uses a customized threshold to judge whether a node is overloaded. If the number of tasks on a certain node is greater than this threshold, then tasks are transferred to the under-loaded ones. The improvement of performance by this method is limited. This paper proposes a global optimization to get a dramatic improvement.

Table 3 Makespan of an optimized schedule

5 tasks on 3 nodes	n_0	n_1	n_2
t_0	-	-	8
t_1	-	8	-
t_2	2	-	-
t_3	3	-	-
t_4	5	-	-

4 Problem definition

Based on the aforementioned motivation, we address the scheduling problem of multiclass tasks with availability constraints in heterogeneous systems. Figure 1 outlines the model of a scheduling mechanism for heterogeneous computing systems. When tasks arrive at the scheduler, the tasks enter a queue and wait for processing by the scheduler, which is in charge of making scheduling decisions and assigning tasks to the appropriate computing nodes for execution.

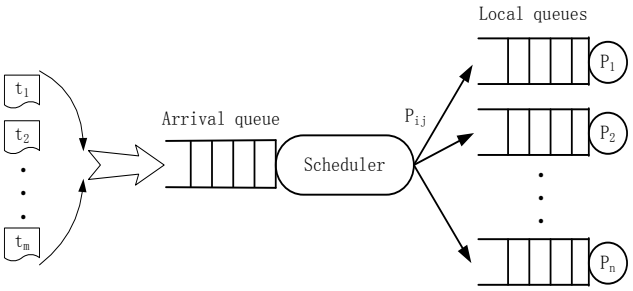

Figure 1 A scheduling mechanism for heterogeneous computing systems

In this model, we consider a heterogeneous system containing n nodes connected by a network to process independent m classes of non-preemptive tasks. Let $N = \{n_1, n_2, ..., n_j, ..., n_n\}$ denotes the set of heterogeneous computing nodes, which have various speeds and availability levels.

Note that the following definitions of makespan and availability refer to those in [7]. And we propose our optimization method based on them in Section 5.

4.1 Makespan analysis

There are m classes of tasks submitted to a heterogeneous system by users. Tasks are independent from each other. Each class of tasks has an availability requirement specified by the user. Values of the availability levels are normalized in the range from 0 to 1. For example, users may set availability level of critical tasks to 1, which means that critical tasks should be assigned to a node which ensures that they can be successfully completed. In this paper, it is also assumed that tasks of the i th ($1 \le i \le m$) class arrive according to a Poisson

process with rate λ_i. All classes of tasks arrive at the system at an aggregated rate $\lambda = \sum_{i=1}^{m} \lambda_i$. Let p_{ij} be the probability that tasks of the i th class are dispatched to node j, where $1 \le j \le n$. Hence, the aggregated task arrival rate of the j th node is expressed as

$$\Lambda_j = \sum_{i=1}^{m} p_{ij} \lambda_i \quad (1)$$

Let μ_{ij} denote the service rate of tasks of class i on node j, and the corresponding expected service time is computed by $1/\mu_{ij}$. It should be noted that the service time of the tasks of class i on node j has a general distribution, which is independent of the arrival processes. Thus, the service utilization for all tasks allocated to node j is as below

$$\phi_j = \sum_{i=1}^{m} (p_{ij} \lambda_i / \mu_{ij}) \quad (2)$$

In this study, each node in the system is modeled as a single M/G/1 queue. Thus, the schedule length of node j is computed as

$$TN_j = E(s_j) + \frac{\Lambda_j E(s_j^2)}{2(1-\phi_j)} \quad (3)$$

Where $E(s_j)$ and $E(s_j^2)$ are the expectation and second moment of the service time. $E(s_j)$ and $E(s_j^2)$ are given

$$E(s_j) = \sum_{i=1}^{m} \left(\frac{p_{ij}\lambda_i}{\Lambda j} \cdot \frac{1}{\mu_{ij}}\right) = \frac{1}{\Lambda_j} \sum_{i=1}^{m} \left(\frac{p_{ij}\lambda_i}{\mu_{ij}}\right)$$

$$E(s_j^2) = \sum_{i=1}^{m} \left(\frac{p_{ij}\lambda_i}{\Lambda_j} \cdot s_{ij}^2\right) = \frac{1}{\Lambda} \sum_{i=1}^{m} (p_{ij}\lambda_i s_{ij}^2)$$

Where s_j is the service time of all task classes on node j, s_j^2 is the square of the service time, and s_{ij}^2 is the square of the service time experienced by tasks of class i on node j.

The expected response time TC_i of tasks of class i can be readily derived from the schedule length of nodes (see Eq.3). Hence, we obtain TC_i as given by Eq.4

$$TC_i = \sum_{j=1}^{n} (p_{ij} \cdot TN_j) \quad (4)$$

Now we derive the mean response time of jobs averaged over all the classes from Eq.4 as

$$T = \sum_{i=1}^{m} \left(\frac{\lambda_i}{\lambda} TC_i\right) = \sum_{i=1}^{m} \left(\frac{\lambda_i}{\lambda} \sum_{j=1}^{n} (p_{ij} \cdot TN_j)\right) \quad (5)$$

To minimize the schedule length without taking availability constraints into account, we have to balance load by evenly distributing the service utilization.

4.2 Availability analysis

The availability of node j is characterized by the probability ξ_j that the node is continuously operational for computation during any random period. The availability of a node is modeled as a function determined by a variety of factors including the node's maintenance status, the number of spare devices dedicated for the node, and the presence or absence of anti-virus software. To determine the unavailable rate θ_j of node j, we used the fuzzy-logic based trust model proposed in [16] to aggregate the multiple factors into a normalized scalar value. Detailed information regarding the trust model can be found in [16].

Our availability model is motivated by the reliability models found in the literature [17, 18]. Let us first introduce the availability cost of class i on node j using Eq.6 as below

$$AC_{ij} = p_{ij} \frac{\theta_j}{\mu_{ij}} \quad (6)$$

Where θ_j is the unavailable rate of node j.

Eq.6 shows that the availability cost of class i on node j is directly proportional to two parameters: (1) the probability that tasks of the i th class are dispatched to node j and (2) the unavailable rate of node j. Note that the unavailable rate used in this study is expressed as Eq.7, where α used in our experiments is 0.1. Eq.7 indicates that the unavailable rate of node j is inversely proportional to the availability of node j. System parameter α must agree with the measurements taken from real systems. Availability ξ_j can be estimated and provided by hardware vendors. Eq.7 just gives one way to calculate unavailable rates for illustration purpose, and it is possible to substitute it by any other unavailable rate model.

$$\theta_j = 1 - \exp(-\alpha(1-\xi_j)) \quad (7)$$

The availability cost AC_i of class i is derived from Eq.6 and Eq.7 as follows

$$AC_i = \sum_{j=1}^{n} AC_{ij} = \sum_{j=1}^{n} P_{ij} \frac{\theta_j}{\mu_{ij}} \quad (8)$$

Based on Eq.8, we can express the availability A_i for class i as Eq.9.

$$A_i = \exp[-AC_i] = \exp[-\sum_{j=1}^{n} p_{ij} \frac{\theta_j}{\mu_{ij}}] \quad (9)$$

Now we calculate the availability A exhibited by the system. The system's availability expressed by Eq.10 is the probability that the system is continuously performing at any random period of time.

$$A = \sum_{i=1}^{m} (\frac{\lambda_i}{\lambda} A_i) = \sum_{i=1}^{m} \{\frac{\lambda_i}{\lambda} \exp[-\sum_{j=1}^{n} p_{ij} \frac{\theta_j}{\mu_{ij}}]\} \quad (10)$$

Eq.10 indicates that in order to enhance the system availability, it is necessary to reduce availability cost substantially expressed by Eq.8.

4.3 Problem definition

The problem of scheduling non-preemptive tasks of m classes on n heterogeneous nodes is considered with various speeds and availability. Although each task can be completed at any node, some nodes may be more efficient than other nodes in terms of processing speed. The goal is to find a schedule that provides a good trade-off between the availability and the makespan.

Let p_{ij} be the probability that tasks of the i th class are dispatched to node j. If $p_{ij} = 1$, it means that task of class i is processed on node j. In our optimization method, p_{ij} is binary, and $p_{ij} \in \{0,1\}$. The optimal strategy is defined as below:

$$P^* \equiv \{P^*_{ij}\}_{1 \le i \le m; 1 \le j \le n} \quad (11)$$

(i) $\forall j, \alpha_i > \xi_j$, which means the availability requirement of the *i*th class of tasks cannot be satisfied by any nodes. The key is to decrease the availability cost of tasks as much as possible without harm to the makespan. Refer to Eq.12.

$$Maxmize \quad A = \sum_{i=1}^{m} \{\frac{\lambda_i}{\lambda} \exp[-\sum_{j=1}^{n} p_{ij} \frac{\theta_j}{\mu_{ij}}]\} \quad (12)$$

(ii) $\exists j, \alpha_i \le \xi_j$, which indicates there is at least one node able to satisfy the availability requirement of the *i*th class of tasks. In this case, the key is to minimize the makespan.

$$Minmize \quad f(x) = \{\max[\sum_{i=1}^{m} p_{i1} TN_{i1}, ..., \sum_{i=1}^{m} p_{in} TN_{in}]\} \quad (13)$$

where $f(x)$ is the makespan.

Eqs.(12) and (13) are constrained by the equation below.

$$s.t \quad \sum_{j=1}^{n} p_{ij} = 1, \forall i \qquad \sum_{i=1}^{m} p_{ij} = 1, \forall j \quad (14)$$

In fact, the problem (ii) has already solved in [7]. But it considers the mean response time there. For the problem (i), the approach in the previous work [7] attaches most attention to the system availability, so that the negative influence on makespan, load imbalance, is caused. Consequently, the following section gives an optimization method to relieve load imbalance and a complete scheduling algorithm based on Ant Colony Optimization (ACO).

5 H2ACO scheduling algorithm

The H2ACO scheduling algorithm contains two key components. The first component is an ant colony optimization algorithm that makes initial scheduling decision. The fitness function is the reciprocal of *f(x)* in Eq.13. The second one is an availability aware scheduling mechanism to optimize the initial schedules made by the first component, thereby improving both availability and performance.

In the following, we outline the H2ACO algorithm that aims at deciding how to map heterogeneous tasks to the most appropriate computing resources to meet users' availability requirements. Our scheme performs the following four steps to make a scheduling decision.

- Step 1: Run a scheduling algorithm to map tasks to set of heterogeneous computing nodes.
- Step 2: Migrate certain tasks from one node to another node which has lower availability cost or satisfies the availability requirement of the task.
- Step 3: Check whether migrations made in Step 2 can satisfy the following condition:
  The total makespan is decreased if doing so.
- Step 4: Repeatedly perform Step 2 and Step 3 until the schedule cannot be further optimized.

Let us denote response times on n nodes as RT_1, RT_2,... RT_n, with which we model its makespan as $RT = \max\{RT_1, RT_2, ..., RT_n\}$. Our optimization scheme aims at improving system availability without adversely increasing the makespan of

schedules. From Eq.6, to enhance system availability actually means to reduce the availability cost.

The conditions used in Steps 2 and 3 to determine the qualified migrations can be formally expressed below.

$$\exists j, \alpha_i \le \xi_j, \text{ Otherwise } \frac{\theta_k}{\mu_{ik}} > \frac{\theta_j}{\mu_{ij}},$$
$$RT_j + TC_{ij} \le RT_{old} \quad (15)$$

where RT_{old} represents the makespan before task i is migrated from node k to node j.

The optimization processes (i.e., Step 2 and Step 3) are repeatedly performed until the schedule cannot be further optimized (see Step 4). After the optimization procedure terminates, the optimization process can fulfill the following conditions.

$$\begin{aligned} A_{new} &> A_{old} \\ RT_{new} &< RT_{old} \end{aligned} \quad (16)$$

where A_{old} and RT_{old} are the old system availability and makespan, while A_{new} and RT_{new} are the new availability and makespan after optimization.

Let us make use of the motivation example to demonstrate how the optimization process is carried out. We assume that the availability cost of t_0 on n_0 is 0.67, while the availability cost of t_0 on node n_2 is 0.54. Table 1 shows that the availability requirement of t_0 is 0.50, which is smaller than the availability of node n_2 (0.60), so the availability of node n_2 can satisfy the availability constraints of t_0. Although the execution time of t_0 on node n_2 is larger than that on node n_0, migrating t_0 to n_2 still can shorten the total makespan (see Table 3).

We can improve the performance by migrating tasks to new nodes after a tentative schedule is made by any immature algorithm. Steps 2-4 give this optimization scheme to further improve the overall scheduling performance.

The H2ACO approach addresses the scheduling problem of m classes of non-preemptive tasks on n heterogeneous nodes with various speed and availability. The initial schedule in Step 1 is generated by an algorithm based on ACO approach. The H2ACO scheme strives to find a schedule that provides a good trade-off between the availability and the makespan. Algorithm 1 outlines the basic idea of H2ACO algorithm, and its time complexity is $O(n+2mn)$.

Algorithm 1 The H2ACO Algorithm

Initialize ants' distribution among nodes
Repeat
  for each ant do
    for each task in an application do
      select next route
    end for
    evaluate fitness of individual path
    update pheromone along its path
  end for
until maximal iterations we set
for $j = 0$ to n **do**
  $RT_{old} = \max\{RT_1, RT_2, ..., RT_n\}$
end for
for $i = 0$ to m **do**
  for $j = 0$ to n **do**
    calculate $\frac{\theta_j}{\mu_{ij}}$
  end for
end for
repeat
  for $i = 0$ to m **do**
    for $j = 0$ to n **do**
      if node j satisfied expression (15) **then**
      move i to j from the original node
      end if
    end for
  end for
until the user terminates the algorithm;

6 Experimental results

To evaluate the efficiency of our algorithm, we implement it in Simgrid - a modular trace-driven simulator, and conduct extensive simulation studies. We also implement H2ACO and deploy it on real clusters to validate our simulation results.

Let us first describe the simulation environment. Simgrid contains a set of core abstractions and functionalities that allow us to quickly implement H2ACO for multiclass applications running in heterogeneous computing environments. Simgrid provides the following two components:

- **The SG toolkit:** is the original low-level toolkit, with which simulations are conducted in the form of explicitly scheduling tasks on some resources.
- **The MSG simulator:** is an application-oriented simulator built using SG. It can be used to perform realistic simulations supported by the SG toolkit. The simulator was built on the concept of communicating agents.

Two performance metrics considered in our experiments are makespan and system availability, which are defined below:

- Makespan: The makespan of a schedule is the maximal time of the individual nodes spent on completing the tasks assigned to them. The scheduling time is equal to the sum of execution time and waiting time.
- Availability: The availability of a heterogeneous system is measured by Equation 10.

Our algorithm is compared with two existing solutions-the SSAC, and ACO algorithms. These algorithms. One is SSAC[7] which have the same model with our approach. The other is the approach of [15] which makes trade-off between makespan and availability using Particle Swarm Optimization (PSO). In the figures, we use PSO to represent the second approach. These algorithms are the best candidate algorithms to be compared with our algorithm.

To make fair comparisons, we implement all the three algorithms within the same scheduling framework. The implementations of the above three algorithms share identical data structures (i.e., an arrival queue and multiple local queues) and supporting modules (e.g., the task dispatcher).

Three groups of experiments are performed. In the first two groups of experiments, we use synthetic task sets running on simulated clusters to show the impacts of the task-set size and service rate on H2ACO and its alternatives. In the last group of experiments, we validate the simulation results by running real applications on a heterogamous cluster.

6.1 Impacts of task sets

In the first group of experiments, we focus on the impacts of the size of task sets on H2ACO and its two alternatives-PSO and SSAC. We schedule and assign 20 tasks on a 5-node heterogeneous cluster. The availabilities of the five computing nodes are 0.804, 0.576, 0.698, 0.732, and 0.468, respectively.

Figures 2-4 shows the availabilities, response times, and makespans of the schedules made by H2ACO, SSAC and PSO under the synthetic task sets when the task size varies from 20 to 100. Figure 2 compares the availabilities of the three algorithms. It decreases when the task size increases. The availability offered by H2ACO is noticeably higher than those provided by the two alternative approaches. For example, when task set is 40, H2ACO substantially improves the system availability.

Figures 3-4 show the response times and makespans of the task sets. For all the three examined algorithms, the response time and makespans are increasing when the task size grows. Makespans largely depend on response time and availability, because makespan measures time interval between the earliest task's arrival time and the last task's finishing time on a specific node. Therefore, the increased makespan is attributed to the increase in response time. Figures 3-4 illustrate that H2ACO helps in reducing both the response time and makespan of schedules. The improvement of H2ACO becomes more pronounced when the task set is large (e.g., 100). H2ACO makes good trade-off between makespan and availability. Thus, the results plotted in Figures 2-4 confirm that compared with SSAC and PSO, H2ACO improves system availability, response time, and makespan in terms of task size.

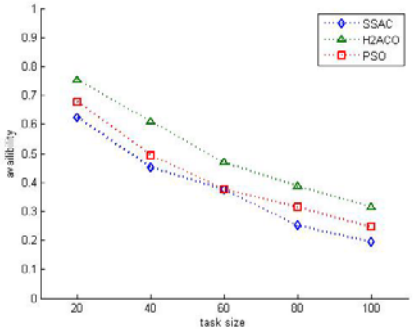

Figure 2 The availability of the schedules made by H2ACO, SSAC and PSO

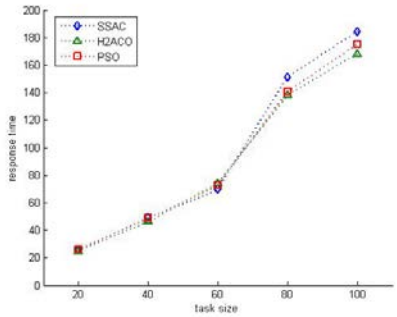

Figure 3 The response time of the schedules made by H2ACO, SSAC and PSO

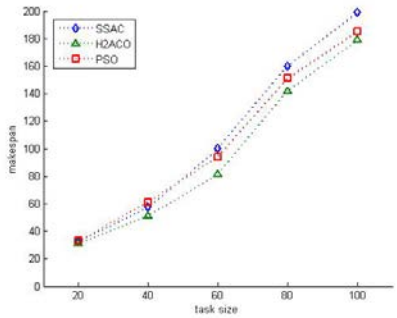

Figure 4 The makespan of the schedules made by H2ACO, SSAC and PSO

6.2 Impacts of service rate

In the second group of experiments, we study the impacts of service rate on H2ACO. Figures 5-6 plot the availabilities and response times of the three scheduling approaches when the service rate

(No./Sec) increases from 0.2 to 1. The size of the task sets in the experiments shown in Figure 5 is 40. The service rate represents the speed at which each node can process tasks. A node with higher service rate handles more tasks than those with lower service rate.

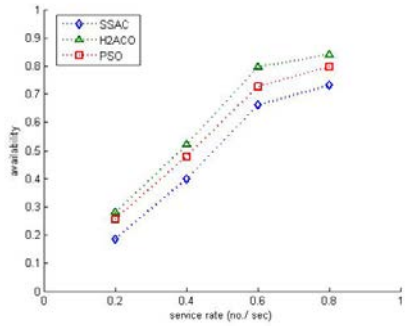

Figure 5 The availabilities of the schedules made by H2ACO, SSAC and PSO

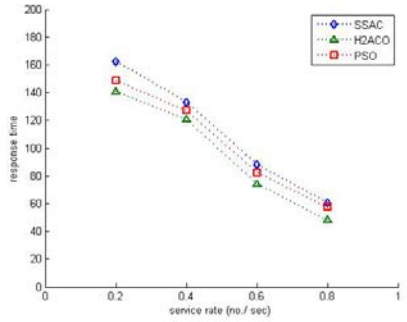

Figure 6 The response times of the schedules made by H2ACO, SSAC and PSO

The results in Figure 5 illustrate that the heterogeneous systems with higher service rates can benefit more from H2ACO in terms of availability. Figure 6 reveals that H2ACO maintains a similar response time in contrast with SSAC and PSO when the service rate is higher than 0.4 No./Sec. When the service rate is low (e.g., 0.2 No./Sec.), H2ACO has the shortest response time compared with SSAC and PSO.

6.3 Real applications and clusters

In the last group of experiments, we compare H2ACO with SSAC and PSO using applications running on 8-node heterogeneous cluster. In this experiment, we run 20 molecular dynamics applications which solve molecular collision problems and simulate the processes of molecular ionization, adsorption, composite, diffusion, and photoionization. To test the scalability of H2ACO, we run the applications on 4-node, 6-node, and 8-node heterogeneous clusters, respectively.

Figures 7-8 shows the availabilities of the schedules made by H2ACO, SSAC and PSO. Figure 7 confirms that for the real scientific applications, H2ACO still offers higher availability than SSAC and PSO. For example, in the case of the 4-node cluster, H2ACO improves the availability over SSAC and PSO by 42.5% and 36.25%, respectively. The 4-node cluster benefits more from H2ACO in terms of availability than 6-node and 8-node clusters, because both SSAC and PSO perform poorly for the 4-node cluster. It is observed that the cluster's availability increases as the number of nodes grows for the three algorithms. This is possibly caused by the reason that more diverse nodes in a heterogeneous cluster offer more opportunities for system availability satisfaction.

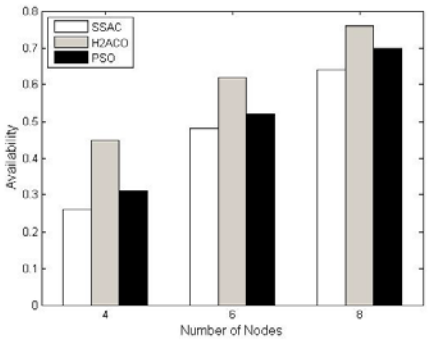

Figure 7 The comparisons of availability measurements among H2ACO, SSAC and PSO

Figure 8 reveals the comparison of makespan measurements among H2ACO, SSAC and PSO. Figure 8 shows that H2ACO achieves the shortest makespan for the real applications compared with both SSAC and PSO. For instance, when it comes to the 8-node cluster running the real applications, H2ACO reduces the makespan by 25.7% and 17.1% in contrast with SSAC and PSO. Moreover, we observe that increasing the number of nodes can help in reducing the makespan. The makespan reduction provided by H2ACO becomes more pronounced when the number of nodes increases from 4 to 8. We conclude that in terms of makespan, large-scale clusters can benefit more from H2ACO than SSAC and PSO.

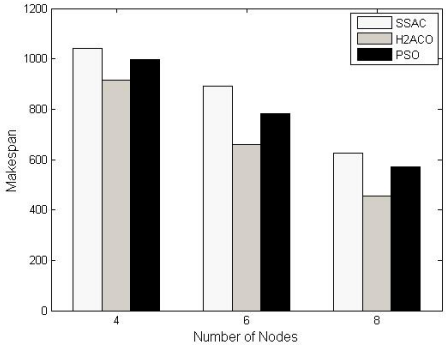

Figure 8 The comparisons of makespan measurements among H2ACO, SSAC and PSO

7 Conclusion and future work

Traditional scheduling solutions can no longer provide satisfactory resource management for heterogeneous systems running multiclass tasks with availability constraints. Based on the existed model, we define the scheduling problem for multiclass applications under heterogeneous computing systems. Then we come up with an optimization mechanism and further propose a scheduling algorithm using Ant Colony Optimization technique. In a word, a better trade-off between availability and makespan is achieved in this study.

In future work, it is planned to extend H2ACO by investigating two intriguing issues: upgrade H2ACO for parallel applications that have flexible availability requirements; apply dynamic programming and branch-and-bound approaches when scheduling multiclass tasks in heterogeneous computing systems.

Acknowledgements This work was supported by the Key Program of National Natural Science Foundation of China (Grant No. 61133005), the Hunan Provincial Natural Science Foundation of China (Grant NO. 13JJ4038) and Youth Growth Plan of Hunan University. Xiao Qin's research was supported by the U.S. National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS).

References

[1] M. Zhu, F. Cao and J. Mi, A hybrid mapping and scheduling algorithm for distributed workflow applications in a heterogeneous computing environment, *Studies in Computational Intelligence*, Vol.382, 2012, pp.117-127.

[2] F. A. Omara, M. M. Arafa, Genetic algorithms for task scheduling problem, *Journal of Parallel and Distributed Computing*, Vol.70, No.1, 2010, pp.13-22.

[3] X. Qin, H. Jiang, A dynamic and reliability-dr -iven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters, *Journal of Parallel and Distributed Computing*, Vol.65, No.8, 2005, pp.885-900.

[4] Y. Lee, A. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, *IEEE Transactions on Parallel and Distributed Systems*, Vol.22, No.8, 2011, pp.1374-1381.

[5] H. Jiang, A. Iyengar and E. Nahum, Design, implementation, and performance of a load balancer for SIP server clusters, *IEEE/ACM Transactions on Networking*, Vol.20, No.4, 2012, pp.1190-1202.

[6] M. A. Salehi, B. Javadi, and R. Buyya, QoS and preemption aware scheduling in federated and virtualized grid computing environments, *Journal of Parallel and Distributed Computing*, Vol.72, No.2, 2012, pp.231-245.

[7] X. Qin, T. Xie, An availability-aware task scheduling strategy for heterogeneous systems, *IEEE Transactions on Computers*, Vol.57, No.2, 2008, pp.188-199.

[8] Schmidt G. Scheduling with limited machine availability. *European Journal of Operational Research*, Vol.121, 1998, pp.1-15.

[9] Apon A. and Wilbur L. Ampneta highly available cluster interconnection network. *In Proceedings of the Parallel and Distributed Processing Symposium*, 2003, pp.1-10.

[10] Sadfi C. and Ouarda Y. Parallel machines scheduling problem with availability constraints. *In Proceedings of the International Workshop on Project Management and Scheduling*, 2004, pp.570-571,.

[11] Wang G., Sun H., and Chu C. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *Annals of Operations Research*, Vol.133, No.4, 2005, pp.183-192.

[12] Kacem I. and Chu C. Worst-case analysis of the wspt and mwspt rules for single machine scheduling with one planned setup period. *European Journal of Operational Research*, Vol.187, 2006, pp.1080-1089.

[13] Kacem I. and Chu C. Efficient branch-and-bound algorithm for minimizing the weighted sum of completion times on a single machine with one availability constraint. *International Journal of Production Economics*, Vol.112, No.1, 2008, pp.138-150.

[14] Jun Zhang, Chris Phillips, Job-scheduling via resource availability prediction for volunteer computational grids, *Int. J. of Grid and Utility Computing*, Vol.2, No.1, 2011, pp.25-32.

[15] Hao Yuan, Yong Wang, and Long Chen, An Availability-Aware Task Scheduling for Heterogeneous Systems Using Quantum-behaved Particle Swarm Optimization, *Lecture Notes in Computer Science*, vol. 6145, 2010, pp.120-127.

[16] S. Song, K. Hwang, and Y. Kwok, Risk-resilient heuristics and genetic algorithms for security -assured grid job scheduling, *Microprocessors and Microsystems*, Vol.55, No.6, 2006, pp. 703-719.

[17] F. Omaraa, M. Arafa, Genetic algorithms for task scheduling problem, *Journal of Parallel and Distributed Computing*, Vol.70, No.1, 2010, pp.13-22.

[18] S. K. Garg, R. Buyya, and H. J. Siegel, Time and cost trade-off management for scheduling parallel applications on utility grids, *Future Generation Computer Systems*, Vol.26, No.8, 2010, pp. 1344-135.

[19] H. Kellerer, V. A. Stusevich, Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications, *Algoirthmica*, Vol.57, No.4, 2010, pp.769-795.