SOFTWARE-PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2014; **44**:163–174
Published online 23 November 2012 in Wiley Online Library (wileyonlinelibrary.com). **DOI**: 10.1002/spe.2163

Bandwidth-aware divisible task scheduling for cloud computing

Weiwei Lin[1,*,†], Chen Liang[1], James Z. Wang[2] and Rajkumar Buyya[3]

[1]*School of Computer Engineering and Science, South China University of Technology, Guangzhou, China*
[2]*School of Computing, Clemson University, PO Box 340974, Clemson, SC 29634-0974, USA*
[3]*Cloud Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Parkville, Victoria Australia*

SUMMARY

Task scheduling is a fundamental issue in achieving high efficiency in cloud computing. However, it is a big challenge for efficient scheduling algorithm design and implementation (as general scheduling problem is NP-complete). Most existing task-scheduling methods of cloud computing only consider task resource requirements for CPU and memory, without considering bandwidth requirements. In order to obtain better performance, in this paper, we propose a bandwidth-aware algorithm for divisible task scheduling in cloud-computing environments. A nonlinear programming model for the divisible task-scheduling problem under the bounded multi-port model is presented. By solving this model, the optimized allocation scheme that determines proper number of tasks assigned to each virtual resource node is obtained. On the basis of the optimized allocation scheme, a heuristic algorithm for divisible load scheduling, called bandwidth-aware task-scheduling (BATS) algorithm, is proposed. The performance of algorithm is evaluated using CloudSim toolkit. Experimental result shows that, compared with the fair-based task-scheduling algorithm, the bandwidth-only task-scheduling algorithm, and the computation-only task-scheduling algorithm, the proposed algorithm (BATS) has better performance. Copyright © 2012 John Wiley & Sons, Ltd.

Received 04 November 2011; Revised 10 July 2012; Accepted 27 September 2012

KEY WORDS: cloud computing; task-scheduling algorithm; nonlinear programming model

1. INTRODUCTION

Nowadays, popular scientific computing applications that need high-performance computing are deployed in large-scale networked computing systems such as Grid. However, grid computing is suitable for specialized application, and grid computing is not available for users all over the world. The new emergence of cloud-computing technologies provides a method to deal with complex applications, which are the applications of great deal of data and which need high-performance applications. As cloud computing has the advantage of delivering a flexible, high-performance, pay-as-you-go, on-demand service offered over the Internet, common users and scientists can use cloud computing to resolve complex applications. Divisible load applications occur in many scientific and engineering applications. Therefore, this paper focuses on studying divisible task scheduling of high-performance computing applications [1–4] in the cloud-computing environment where multiple virtual machines (VMs) can share physical resources (CPU, memory, and bandwidth) on a single physical host, and multiple VMs can share the bandwidth of data center by using network virtualization. Cloud computing has been flourishing in past years because of its ability to provide users with on-demand, flexible, reliable, and low-cost services [5]. Because system resources are essentially shared by many users and applications, an excellent task-scheduling scheme is critical to resource utilization and system performance. Many system parameters, such as processor

*Correspondence to: Weiwei Lin, School of Computer Engineering and Science, South China University of Technology, Guangzhou, China.
†E-mail: linweiwei2004@yahoo.com.cn

power, memory space, and network bandwidth, affect the efficiency of task scheduling. In addition, the heterogeneity of computing sources in different nodes adds the complexity of task scheduling. Furthermore, frequent data exchange among nodes, hosts, and clusters in data-intensive cloud applications makes the task-scheduling procedure extremely complicated. Recently, different models and algorithms [1,6–11] were proposed to deal with task scheduling in a cloud-computing environment. Most of these methods focus on allocating CPU and memory resources to various cloud-computing tasks, assuming that all physical nodes and VMs have unlimited network bandwidth.

Wang *et al.* [6] proposed a multi-dimensional task-scheduling algorithm based on the availability of CPU, memory, and VMs. This algorithm considers the limitation of both physical and virtual resources, and allocates resources according to task needs and resource loads. However, this algorithm did not consider the bandwidth requirements of tasks, nor did it consider the dynamic change of their resource requirements. In [7], a two-level task-scheduling mechanism based on load balancing of resources is presented. The first-level scheduling focuses on assigning user applications to VMs by creating a task description for each VM. Then, at the second level, it allocates proper host resources to VMs on the basis of certain rules. This two-level scheduling scheme also included a strategy to migrate VMs among physical hosts on the basis of loads of VMs to achieve load balance. However, this scheduling algorithm did not consider the impact of network bandwidth and the cost of resource usage. Grounds *et al.* [8] proposed a cost-minimizing scheduling algorithm to address the task-scheduling problem in an application workflow under cloud-computing environment and demonstrated that it outperforms the proportional least laxity first algorithm [9]. However, both of these algorithms did not consider the limitation of network bandwidth. In [10], a nonlinear programming model was proposed to minimize the cost of task execution and data transmission for a data-intensive application workflow. Experiments with an intrusion-detection application workflow showed that this new model outperformed Amazon CloudFront's 'nearest' single data source selection approach. Li [11] proposed a task-scheduling algorithm based on a non-preemptive priority M/G/1 queuing model to meet the QoS requirements of cloud-computing users. In [1], an on-line dynamic task-scheduling algorithm, scheduling algorithm based on load-balance and availability fuzzy prediction (SALAF), was proposed on the basis of a two-objective model and fuzzy prediction method. Experimental results showed that the SALAF algorithm is better than both min–min algorithm [12] and suffrage algorithm [13]. In [2], a double-fitness genetic algorithm was proposed for the programming framework of cloud computing. It shortens not only the total task completion time but also the average completion time. In [3], a performance analysis of cloud-computing services for the many-task scientific computing is presented. Experimental results indicated that the computation performance of the tested clouds is low, and the current clouds need an order of magnitude in performance improvement to be useful to the scientific community.

Nonetheless, these task-scheduling algorithms did not consider the impact of network bandwidth. However, tasks scheduled based only on the availability of CPU and memory resources may be delayed because of insufficient network bandwidth, resulting in waste of CPU and memory resources. With these task-scheduling algorithms, service providers of the cloud computing often have to upgrade hardware to meet the demand of network bandwidth. Hardware upgrades cost more to service providers, defeat the intention of cloud-computing services, and are not scalable [14]. On the other hand, some studies focus on allocating network bandwidth to meet the task requirements without considering the limitation of CPU and memory resources. A similar work has been done in heterogeneous computing systems [15]. But it is not for cloud computing. Sonnek *et al.* [16] presented a decentralized affinity-aware migration technique that incorporates heterogeneity and dynamism in network topology and job communication patterns to allocate VMs on the available physical resources. Wang *et al.* [17] proposed a dynamic task-scheduling algorithm by solving minimum congestion model to avoid sharing of the same network link by multiple resources. In [18], a multi-port, bandwidth-bounded model was proposed for task scheduling on heterogeneous computing platform. However, the authors showed that scheduling divisible load under the bounded multi-port model is NP-complete even if only CPU and memory resources are considered. As finding an optimal task schedule for cloud computing with only CPU and memory resources considered is an NP-complete problem, adding network bandwidth into the mix will further increase the complexity.

In this paper, we propose a nonlinear programming method to solve the bounded multi-port model and, in turn, design a new task-scheduling algorithm, the bandwidth-aware task-scheduling (BATS) algorithm, to allocate the proper number of tasks to each VM on the basis of its CPU power, memory space, and network bandwidth. In contrast, most existing task-scheduling algorithms for cloud computing only considered the availability of CPU and memory resources in task scheduling. A few of them, on the other hand, focused on the limitation of network bandwidth in task scheduling. None of them sought to optimize the utilization of all three system resources.

The rest of this paper is organized as follows. We present the nonlinear programming for solving the multi-port, bandwidth-bounded model in Section 2 and the BATS algorithm in Section 3. In Section 4, we discuss the implementation of the BATS algorithm under the CloudSim environment. Then, in Section 5, we conduct a performance evaluation using the CloudSim toolkit. After discussing the experimental results, we give our conclusion in Section 6.

2. NONLINEAR PROGRAMMING MODEL FOR TASK SCHEDULING

As shown in Figure 1, assume that a cloud-computing platform is supported by m physical computer hosts, $PH_1, \ldots, PH_m$, each of which hosts a set of VMs with the corresponding virtual machine monitor (VMM). We adapt the bounded bandwidth multi-port communication model that was originally designed for heterogeneous distributed computing environment [19] to schedule independent tasks on cloud-computing platforms. In this multi-port cloud-computing task-scheduling model with bounded bandwidth (as shown in Figure 2), there are n virtual machines $p_1,p_2,\ldots,p_n$ with computing power (time needed to compute a unit task) $w_1,w_2,\ldots,w_n$, and the scheduling node p_0 that can serve any number of VMs simultaneously, and each VM has an incoming bandwidth b_i, $1 \le i \le n$. As the cloud-computing platform itself has a limited network bandwidth, which is shared by all VMs, we

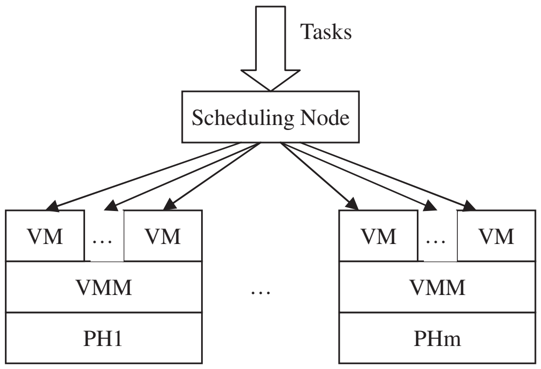

Figure 1. Framework of task scheduling on the cloud platform.

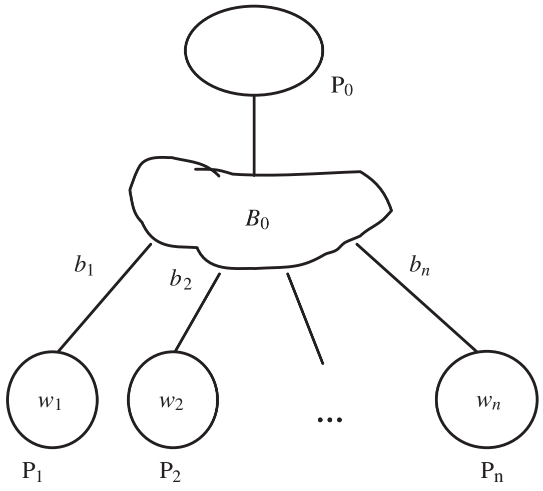

Figure 2. Bandwidth-bounded multi-port task-scheduling model in cloud computing.

can assume that the cloud-computing platform is associated to an outgoing bandwidth B_0, as we know that the dynamic bandwidth allocated for VMs [17,20,21] in cloud computing can improve the performance of the system. Each VM is allocated a dynamic bandwidth $b'_i \le b_i$, but the total outgoing bandwidth of all the VMs is not exceeded, that is, $\sum b'_i \le B_0$.

There are M equal-sized independent tasks to be scheduled under the bandwidth-bounded cloud-computing environment. In order to minimize the total time (T) needed to finish all tasks, the scheduling node P_0 needs to decide how many tasks should be specifically allocated and transferred to each VM. In order to achieve the optimized task allocation, we formulate a nonlinear programming model for the task-scheduling problem with the following constraints:

$$\begin{cases} \text{Minimize } T \\ \text{Subject to :} \\ (1)\sum_{i=1}^{n} x_i = M \\ (2) 0 \le x_i \le M \ \text{ for } 1 \le i \le n \\ (3) 0 \le w_i, \ 0 \le b_i \ \text{ for } 1 \le i \le n \\ (4) w'_i = \max(w_i, 1/b_i) \\ (5) w'_i \cdot x_i \le T - 1/b_i \ \text{ for } 1 \le i \le n, x_i \ge 1 \\ (6) \sum_{i=1}^{n} (x_i/(T - w_i)) \le B_0 \ \text{ for } 1 \le i \le n \\ (7) M, n, x_i \ \text{ are all integers} \end{cases}$$

1. Ensures that the total number of tasks executed by all nodes is equal to M, where x_i denotes the number of tasks allocated to node P_i.
2. Ensures that the number of tasks allocated to each of VMs is less than or equal to M.
3. Ensures that each VM node has computing power and bandwidth greater than 0.
4. Ensures that the execution of algorithm will not be misled and will not violate the real situation when task execution time w_i is less than the task transmission time $1/b_i$. In such case, there will certainly exist the state that the previous task is already finished; however, the next task is yet to come. When that happens, VM could do nothing but wait. As such waiting unavoidably exists, the way to handle the waiting in the model is to append the time difference to execution time. In other words, as the execution waits until transmission ends, the scheduling strategy regards execution time w_i the same as transmission time $1/b_i$. The preceding equation is the case when w_i is less than $1/ b_i$; otherwise, that is, w_i is greater than or equal to $1/ b_i$, the time difference mentioned earlier does not exist, and no change to w_i is needed. On the basis of the earlier discussion, we can reach the following constraint: If w_i is less than $1/b_i$, then let $w_i = 1/b_i$; otherwise, w_i remains unchanged. Thus, it is $w'_i = \max(w_i, 1/b_i)$.
5. Ensures that the time of each node P_i spent on computing tasks must be less than or equal to the total time of finishing all tasks (T) minus the start time of computation (the communication time of transferring a single task from the scheduling node to the VM node P_i).
6. Ensures that the scheduling node could communicate with multiple nodes simultaneously, and the sum of bandwidth in transferring tasks must be less than or equal to the upper bound B_0, which means $\sum b'_i \le B_0$.

When the number of VMs connected to scheduling node is not very large (the variable n is not very large), the solution to the nonlinear programming model could be obtained in a short period by using DONLP2 [22] or LINGO [23]. The solution is the optimized task allocation scheme that determines the proper number of tasks assigned to each virtual resource node. On the basis of the optimized allocation scheme, we can design a heuristic task-scheduling algorithm.

3. DESIGN OF BATS

On the basis of the earlier discussion, we can easily establish a nonlinear programming model for task scheduling in bandwidth-bounded cloud computing. By solving the nonlinear programming model, we can obtain the optimized task allocation scheme, which gives a proper number of tasks on each VM.

An efficient heuristic algorithm for task scheduling can be built with the optimized task allocation scheme as heuristic information. We name the heuristic algorithm on the basis of the optimized task allocation scheme, BATS. In our simulation, we have the concept of Broker, which works on behalf of users; all the details are related to the VM creation, the VM destruction, and task submission to these VMs. In CloudSim, Broker is a Java object responsible for accepting user's requests and creates a schedule for completing user's requests. It accepts a list of VMs and a list of cloudlets to be completed from user, and then it is responsible for binding all cloudlets to these VMs in a certain way and submitting all cloudlets. Eventually, it will return the result of execution to user when all cloudlets are done. The algorithm is deployed on Broker and works out the optimized task allocation scheme on the basis of information of VM (computing power and bandwidth) and submits the optimized scheme to the Datacenter, so that the Datacenter allocates tasks to VMs according to the optimized task allocation scheme. In BATS, if a set of tasks that represent a divisible load application is submitted to the Datacenter, then obtain computing power and bandwidth of VMs owned by Broker, obtain information about tasks, build a nonlinear programming model, and solve the model and obtain the optimized task allocation scheme A that determines proper number of tasks assigned to each VM. In each iteration, one VM is bound the proper number of tasks. Then, the Broker sets bandwidth used in task transmission and transfers task on the basis of the bandwidth. The details of the pseudo code of BATS is depicted as follows:

```
Procedure BATS ()//scheduling algorithm of Broker
    If (Task_Submitted())//if there is a set of tasks submitted
        GetVmInfo()//get computing power and bandwidth of VMs owned by Broker
        GetTaskInfo()//get information about tasks, including total number and size
        Build_Model(model)//build a nonlinear programming model
        A=Solve(model)//solve the model and obtain the optimized task allocation scheme A
    End If
    While (Exist_Idle())//when there is a idle VM in virtual machine list vmlist
       vmi=GetNextIdle(vmlist)//get next idle virtual machine vmi from vmlist
       Ti=GetTaskNum(A)//get the number of tasks, Ti, allocated to vmi in A
       Bind(A, vmi)//bind Ti tasks to vmi
       SetBandwidth(vmi, b&prime;i)// set bandwidth used in task transmission: b&prime;i=xi/(T-wi)
    End while
    Submit ()//concurrently send tasks with new bandwidth
End if
```

The time complexity of BATS is computed for two phases (solve-time and allocate-time). In the allocate-time, BATS must allocate tasks to each VM. Thus, the time complexity of the allocate-time phase of the algorithm is $O(n)$, where n is the number of VMs. In the solve-time phase, it needs to solve the nonlinear programming model, which is proposed to obtain the proper number of tasks assigned to each VM in Section 2. The time complexity of the scheduling algorithm directly does not depend on the size of tasks. Therefore, when we are scheduling a very large number of tasks, the extra time overhead produced by BATS is insignificant as compared with the time needed to finish the tasks. Furthermore, we observe that some VMs may be equivalent, in terms of their computing power and bandwidth, and try to exploit this information to reduce the number of the variable x_i and improve the solving of the nonlinear programming model. We used LINGO to solve the model. The computation time of the solver to reach a solution (for the situation of 200 VMs and 10,000 tasks) was less than 60 s, which is insignificant as compared with the task completion time in our experiments.

4. CloudSim-BASED SIMULATION OF BATS

In order to evaluate the proposed model and algorithm, we have simulated the proposed task-scheduling algorithm, and infrastructure and application scenarios with CloudSim [24]. To implement the algorithm, we need to overwrite the class DatacenterBroker in CloudSim. Specifically speaking, we need to make DatacenterBroker have the ability of obtaining computing power and bandwidth of VMs and solve the nonlinear programming model by using Lingo. The program acquires computing power and bandwidth as input parameters, and output the number of tasks allocated to each VM as variables, along with the goal of minimizing *T*, which is the total time of finishing all tasks.

On the basis of this implementation, in which Lingo is called by CloudSim to model and solve the problem, we can extend class DatacenterBroker as follows. Add method bindAll(), it acquires computing power and bandwidth of all VMs currently owned by the user/broker, and then use Lingo's Java interface to construct and solve the nonlinear programming model. In order to implement this process, changes on class DatacenterBroker in CloudSim are needed. When user submits tasks, DatacenterBroker first obtains parameters of VMs currently owned by the Broker, including computing power and communication bandwidth. DatacenterBroker then passes as input parameters all these information, along with total number and size of tasks to Lingo, starts Lingo, and blocks until the result of execution is returned from Lingo. Such result is a set of integers that refer to number of tasks allocated to each VM. Allocating tasks according to these numbers could result in the minimum total time of execution. At last, we only need to call the method bindAll() after the submission of user's task list. The system will automatically generate the optimized task allocation scheme on the basis of parameters of VMs currently owned by the user and distribute tasks following the optimal scheme. The extended data center class, myDatacenterBroker, can be presented as follows:

```
public class myDatacenterBroker extends DatacenterBroker{
broker.submitCloudletList(cloudletList);
broker.bindAll();
. . .
}
```

As mentioned earlier, the following method bindAdd() is added into class myDatacenterBroker. This method calls Lingo to solve the model and obtains the optimized task allocation scheme, then allocates tasks to VMs according to the optimized scheme.

```
 public void bindAll(){
 . . .
/*create object of lingoSolver which implements communication between CloudSim and
Lingo*/
 lingoSolver ls = new lingoSolver();
/*pass parameters, start solving and get the result*/
 double[] res = ls.Solve(num_vm,M,B,length, mips, bandwidth);
 . . .
/*get the optimized allocation scheme*/
for(int v=0;v<x.length;v++)
   x[v]=res[v+1];
 . . .
/*implement the allocation scheme by binding tasks to specific VM*/
int count=0;
for(int i=0;i<num_vm;i++){
   for(int b=count;b<count+lets[i];b++){
      this.getCloudletList().get(b).setVmId(i);
   }
   count+=lets[i];
 }
 . . .
}
```

To manage the communication between CloudSim and Lingo, class lingoSolver is added. The main method of this class is Solve(int N,int M,int B,long length,double[] mips,long[] bw). It obtains information needed for decision making of scheduling. Such information refers to computing power, communication bandwidth, size, and total number of tasks. The Solve() method as well creates and initializes necessary environment for calling Lingo. Then the method passes parameters to Lingo and obtains the optimized task allocation scheme. The LSsetPointerLng method of Lingo is used to write input parameter and retrieve optimized scheme from Lingo. We use this function to pass parameters of VMs to Lingo and use its built-in language for solving.

```
    public class lingoSolver{
    . . .

      /*Solve method gets parameters and passes to Lingo, then waits for result of computation;
    N is the number of physical machine, M is the number of virtual machine, B is the upper bound for
bandwidth, length is the size of each cloudlet, mips is the computing capability requested by each virtual
machine, bw is the bandwidth requested by each virtual machine. */
  public double[] Solve(int N,int M,int B,long length,double[] mips,long[] bw){
    . . .
    /*pass basic information to Lingo, including number and size of tasks*/
    nErr = lng.LSsetPointerLng( pnLngEnv, DATA, nPointersNow);
    /*pass computing power to Lingo*/
    int[] nPointersNow = new int[1];
    nErr = lng.LSsetPointerLng( pnLngEnv, mips, nPointersNow); . . .
    /*pass bandwidth to Lingo*/
    nErr = lng.LSsetPointerLng( pnLngEnv, bandwidth, nPointersNow); . . .
    /*obtain the task allocation scheme and store it in array x*/
    nErr = lng.LSsetPointerLng( pnLngEnv, x, nPointersNow); . . .
    /*get the minimum value of T which is the total time needed to finish all tasks*/
    nErr = lng.LSsetPointerLng( pnLngEnv, T, nPointersNow); . . .
    }
  }
```

Class lingoSolver calls Lingo to build and solve the model through script. Thus, we need to make a file using Lingo's built-in language to solve the problem and build the model with specific objective function:

```
!constraint 1;
s1 = @sum(xx(i):x);
s1 = M;
!constraint 2,7;
@for(xx(i):@gin(x(i)));
!constraint 4;
@for(mmips(i):nmips(i)=@smin(mips(i),bw(i)));
!constraint 5;
@for(bbw(i):Dat(3)/nmips(i)*x(i)<=T-Dat(3)/bw(i));
!constraint 6;
@for(bbw(i):nbw(i) = L*x(i)/(T-L/mips(i)));
s2 = @sum(nnbw(i):nbw);
s2<=B0;
!objective function;
min = T;
```

5. PERFORMANCE EVALUATION

5.1. Experiments and results

We compared BATS with three other algorithms: FBTS, COTS, and BOTS. We conducted experiments with four scheduling algorithms under the same condition and compare the performance of the four algorithms. Among the four algorithms, FBTS [24,25] is the one that does not consider the difference of computing power and bandwidth between different VMs. In such case, the system evenly allocates tasks to each VM, regardless of its computing power and bandwidth. The algorithm COTS [1,26] only takes into account the computing power but does not consider the influence of bandwidth and allocate tasks entirely depending on computing power. Similarly, BOTS [27,26] only takes into account bandwidth when scheduling tasks.

In order to analyze the performance of the algorithm in different scale experiments, we simulated three cloud-computing infrastructure scenarios, with CloudSim. The first two scenarios have five VMs and 25 VMs, respectively, and the last scenario has 200 VMs, each with a heterogeneous computing power and bandwidth. The application simulated in our experiments embodies a divisible workload model [18,28] whose task sizes are equal. We conducted the task-scheduling experiments for computing a large set of equal-sized independent tasks, which is similar to the work in [19,29], on the three infrastructure scenarios.

Experiment 1

Computing power and bandwidth of five VMs are as shown in Table 1. Each of these values is randomly selected from 1000, 2000, 3000, 4000, 5000, and 6000. This implies that computing power and bandwidth of all VMs are in the same order of magnitude.

One hundred tasks, the size of each task is 60,000 MIs [24], are scheduled in this experiment. By specifically putting the aforementioned data into the four algorithms, we obtained Figure 3 showing the total time needed to complete all tasks by the four algorithms.

In this experiment, the time needed for BATS to finish all tasks is 710 s, time needed for FBTS is 1260 s, the time needed for COTS is 1260 s, and the time needed for BOTS is 810 s. As the new algorithm allocates tasks according to the computing power and bandwidth of VMs, specifically compared with other three algorithms, it saves 43.65%, 43.65%, and 12.35% of time.

Experiment 2

In this experiment, we created 25 VMs, each with a heterogeneous computing and bandwidth. One thousand tasks were submitted to Datacenter and scheduled respectively using the four algorithms. The total times needed in the four algorithms are put into comparison. The extra cost brought by BATS is also computed.

Computing power and bandwidth of 25 VMs are as shown in Table 2. Each of these values is randomly selected from 1000, 2000, 3000, 4000, 5000, and 6000. This implies that computing power and bandwidth of all VMs are in the same order of magnitude.

The size of task is 60,000 MIs. By specifically putting the aforementioned data into the four algorithms, we can obtain Figure 4 showing total time needed to complete all tasks by the four algorithms.

Table I. VMs' resource settings.

Virtual machine	Computing power (MIPS)	Bandwidth (Mbps)
VM#1	3000	3000
VM#2	1000	1000
VM#3	1000	4000
VM#4	3000	6000
VM#5	2000	1000

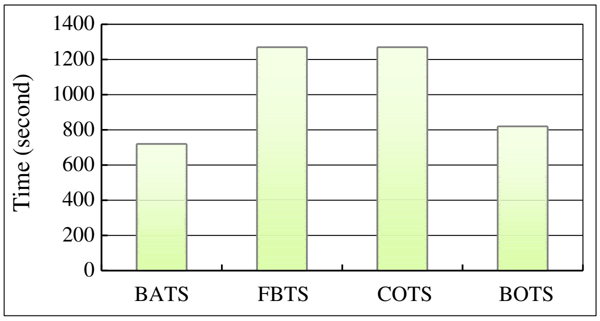

Figure 3. Total time needed to complete all tasks by the four algorithms.

Table II. VMs' resource settings.

Virtual machine	Computing power (MIPS)	Bandwidth (Mbps)	Virtual machine	Computing power (MIPS)	Bandwidth (Mbps)
VM#1	4000	1000	VM#14	3000	2000
VM#2	2000	3000	VM#15	3000	6000
VM#3	4000	5000	VM#16	6000	4000
VM#4	5000	6000	VM#17	1000	4000
VM#5	4000	4000	VM#18	3000	3000
VM#6	1000	2000	VM#19	6000	6000
VM#7	3000	5000	VM#20	3000	5000
VM#8	1000	2000	VM#21	5000	6000
VM#9	5000	3000	VM#22	3000	1000
VM#10	3000	5000	VM#23	3000	6000
VM#11	6000	3000	VM#24	4000	3000
VM#12	5000	4000	VM#25	2000	1000
VM#13	1000	6000			

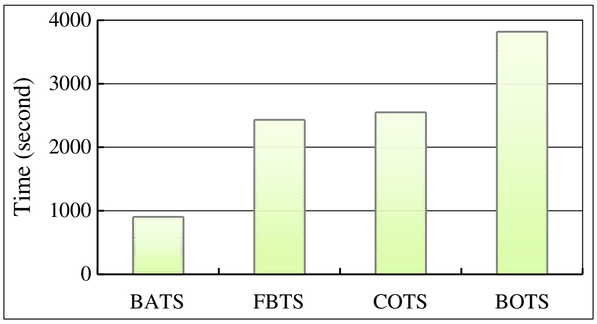

Figure 4. Total time needed to complete all tasks by the four algorithms.

In this experiment, the time needed for new algorithm BATS to finish all tasks is 872 s, the time needed for CloudSim's default algorithm FBTS is 2403 s, the time needed for COTS is 2520 s, and the time needed for BOTS is 3790 s. Specifically compared with other three algorithms, new algorithm saves 63.63%, 65.32%, and 76.94% of time.

Experiment 3

The two experiments above work on 5 VMs and 25 VMs respectively, yet a much larger number of VMs and tasks are generally involved in practical use. Therefore, to further verify our algorithm and analyze its time complexity, we need to experiment with larger scale of input data, for it is more realistic.

In this experiment, we created a Cloud-based data center that has 200 VMs, each with a heterogeneous computing and bandwidth. Ten thousand tasks were submitted to Datacenter and scheduled respectively using the four algorithms. The total times needed in the four algorithms are put into comparison. The extra cost brought by BATS is also computed.

Computing power and bandwidth of 200 VMs are as shown in Figure 5. Each of these values is randomly selected from 1000, 2000, 3000, 4000, 5000, and 6000. This implies that some VMs may be equivalent, in terms of their computing power and bandwidth. We observe that there are about 30 different types of VMs.

The size of task is also 60,000 MIs. By specifically putting aforementioned data into the four algorithms, we can obtain Figure 5 showing total time needed to complete all tasks by the four algorithms.

In this experiment, the time needed for new algorithm BATS to finish all tasks is 2056 s, the time needed for CloudSim's default algorithm FBTS is 3247 s, the time needed for COTS is 3318 s, and the time needed for BOTS is 4642 s. Specifically compared with other three algorithms, new algorithm saves 36.68%, 38.03%, and 55.71% of time.

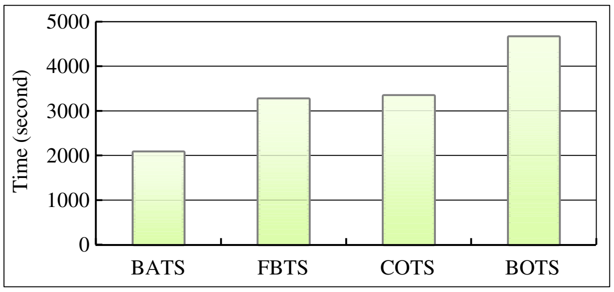

Figure 5. Total time needed to complete all tasks by the four algorithms.

5.2. Analysis on cost of BATS

From the aforementioned experiments, we can see that task allocation scheme generated by BATS efficiently reduces time needed to finish tasks. However, as BATS needs to build and solve the nonlinear programming model to find optimized task allocation scheme, it brings extra cost of time. We used Lingo to solve the model. In the first two experiments, 100 tasks are allocated on five VMs, about 1 s extra time is brought by BATS. However, as the total time of finishing all tasks is greatly reduced, the overall performance is still efficiently improved. In the third experiment, 10,000 tasks were allocated to 200 VMs, the extra time overhead produced by BATS is about 60 s, which is insignificant as compared with the time needed to finish the tasks in our experiments. By cutting the time of executing tasks, BATS brings a large improvement on performance. Although new overhead is generated, BATS could still acquire the minimum duration of executing tasks.

5.3. Discussion

From the experimental results discussed earlier, we can reach the conclusion that the proposed algorithm BATS is better than the other three. The more different the capability of VMs is, the more time new algorithm saves. It is because when the difference between VMs is huge, and if tasks are still allocated evenly, although the VMs with better performance could finish work very soon, the other VMs with lower computing and communication speed would still run for a longer time. Thus, the former will be idle until the latter finishes working. That results in waste of computing resource. Similar problem exists in the other two algorithms. To COTS, as bandwidth is ignored, if the communication speed is less than computation speed, then there will be the status in which the previous task is finished, but the next one has not yet come. The VM will have to wait in this case. To BOTS, it is highly possible that the communication finishes very soon, but tasks have to be put into the waiting list for the sake of the low computation speed. The new algorithm BOTS tries to ensure that every VM keeps working during the entire scheduling, and it, at the same time, reduces waiting as much as possible, especially when the performance gap between VMs is huge. It greatly improves utilization ratio of computing resource.

To sum up, currently, many general methods do not take into account the difference on performance of VMs, and allocate tasks to each VM evenly. This impedes the advancement of system performance. Particularly, when the difference of VM is remarkable, such allocation strategy causes a large waste of computing resource. The improved method introduces VMs computing power and bandwidth into scheduling, and model the problem as a nonlinear programming problem. By finding and applying the optimized allocation scheme, the new algorithm BATS improves utilization ratio of computing resource and reduces execution time. When BATS is used for scheduling high-performance computing applications in the cloud data center, it can improve the efficiency of task scheduling and resource utilization. Thus, this algorithm can help the cloud data center to save cost and to increase revenue.

6. CONCLUSIONS AND FUTURE WORK

In this work, we focus on divisible load applications, where an application load can be divided into a number of tasks that can be processed independently in parallel [18,28]. In order to obtain minimum execution time, we modeled the scheduling problem as a nonlinear programming model. The model can optimize divisible task allocation on the basis of task resource requirements for computation and communication. Using Lingo to solve the model, we can obtain the optimized task allocation scheme. We also designed a heuristic algorithm for divisible task scheduling (BATS) on the basis of the scheme and implemented it with CloudSim. Results of experiment show that compared with FBTS, COTS, and BOTS, BATS can achieve a better performance. The algorithm BATS is ideal for the task scheduling in bandwidth-bounded cloud-computing environments.

As a future work, we plan to extend our model to a situation that there are multiple data centers. In addition, the recent studies have revealed that data centers consume an unprecedented amount of electrical power; hence, we aim to improve task-scheduling optimization and make optimization policy to optimize not only the efficiency but also the energy. We also plan to implement our algorithm in the Xen VMM.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions that have improved the presentation, quality, and correctness of this paper. This work is financially supported by the National Natural Science Foundation of China (no. 61070015) , Guangdong Natural Science Foundation (no. 10451064101005155, S2012030006242 and S2011010001754), GuangZhou City Science and Technology Projects (no. x2jsB2120750), Guangdong Provincial Science and technology projects (no. 2012B010100030) and Guangdong Province strategic emerging industry project (no. 2011A010801002). James Wang's work is partially supported by NSF grant DBI-0960586.

REFERENCES

1. Kong X, Lin C, Jiang Y, Yan W, Chu X-W. Efficient dynamic task scheduling in virtualized data centers with fuzzy prediction. *Journal of Network and Computer Applications* 2011; **34**(4): 1068–1077.
2. Guo L, Zhao S, Shen S, Jiang C. Task scheduling optimization in cloud computing based on heuristic algorithm. *Journal of Networks* 2012; **7**(3): 547–553.
3. Iosup A, Ostermann S, Yigitbasi N, *et al*. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems* 2011; **22**(6): 931–945.
4. Jackson KR, Muriki K, *et al*. Performance and cost analysis of the Supernova factory on the Amazon AWS cloud. *Scientific Programming* 2011; **19**(2–3): 107–119.
5. Buyya R, Yeo CS, Venugopal S. Market oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. *Proceeding of the 10th IEEE International Conference on the High Performance Computing and Communications*, 2008.
6. Wang L, von Laszewski G, Kunze M, Tao J. Schedule distributed virtual machines in a service oriented environment. *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010; 230–236.
7. Fang Y, Wang F, Ge J. A task scheduling algorithm based on load balancing in cloud computing. *Web Information Systems and Mining, Lecture Notes in Computer Science* 2010; **6318**: 271–277.
8. Grounds NG, Antonio JK, Muehring JT. Cost-minimizing scheduling of workflows on a cloud of memory managed multicore machines. *Proceedings of the 1st International Conference on Cloud Computing, Lecture Notes in Computer Science*, vol. **5931**, 2009; 435–450.
9. Shrestha HK, Grounds N, Madden J, Martin M, Antonio JK, Sachs J, Zuech J, Sanchez C. Scheduling workflows on a cluster of memory managed multicore machines. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 2009.
10. Pandey S, Barker A, Gupta KK, Buyya R. Minimizing execution costs when using globally distributed cloud services. *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010; 222–229.
11. Luqun L. An optimistic differentiated service job scheduling system for cloud computing service users and providers. *Proceedings of the third International Conference on Multimedia and Ubiquitous Engineering*, 2009; 295–299.
12. Qin X, Xie T. An availability-aware task scheduling strategy for heterogeneous systems. *IEEE Transactions on Computers* 2008; **57**(2): 188–199.
13. Song S, Hwang K, Kwok Y. Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling. *IEEE Transactions on Computers* 2006; **55**(6): 703–719.
14. Sjaugi MF, Othman M, Napis Suhaimi. Review on concurrent data transfer in grid computing. *Managed Grids and Cloud Systems in the Asia-Pacific Research Community* 2010; **Part 3**: 195–202. DOI: 10.1007/978-1-4419-6469-4_14.
15. Hong B, Prasanna VK. Bandwidth-aware resource allocation for computing independent tasks in heterogeneous computing systems. *15th Annual International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, November, 2003.

16. Sonnek JD, Greensky JBSG, Reutiman R, Chandra A. Starling: minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. *Proceedings of the 39th International Conference on Parallel Processing*, 2010; 228–237.
17. Wang C, Wang C, Yuan Y. Dynamic bandwidth allocation for preventing congestion in Datacenter networks. *Proceedings of the 8th International Symposium on Neural Networks*, Guilin, China, May 29–June 1, 2011.
18. Beaumont O, Bonichon N, Eyraud-Dubois L. Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model. *Proceeding of the 22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008*, Miami, Florida USA, April 14–18, 2008.
19. Hong B, Prasanna VK. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. *Proceedings of the 18th International Proceedings of Parallel and Distributed Processing Symposium (IPDPS'04)*, April 2004; 52–60.
20. Mathew K, Kulkarni P, Apte V. Network bandwidth configuration tool for Xen virtual machines. *Proceedings of the 2nd International Conference on Communication Systems and Networks (COMSNETS'10)*, 2010; 1–2.
21. Mehdi N, Mamat A, Ibrahim H, Symban S. Virtual machines cooperation for impatient jobs under cloud paradigm. *International Journal of Information and Communication Engineering* 2011; **7**(1): 13–19.
22. Spellucci P. A SQP method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming* 1993; **82**(3): 413–448.
23. Farkas T, Czuczai B, Rev E, Lelkes Z. New MINLP model and modified outer approximation algorithm for distillation column synthesis. *Industrial and Engineering Chemistry Research* 2008; **47**: 3088–3103.
24. Calheiros R, Ranjan R, Beloglazov A, Rose C, Buyya R. *CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms*. Software: Practice and Experience (SPE), vol. **41**(1), ISSN: 0038-0644. Wiley Press: New York, NY, January, 2011; 23–50.
25. Jayarani R, Vasanth Ram R, Sadhasivam S, Nagaveni N. Design and implementation of an efficient two-level scheduler for cloud computing environment. *Proceedings of 2009 International Conference on Advances in Recent Technologies in Communication and Computing*, 2009; 884–886.
26. Shi W, Hong B. Resource allocation with a budget constraint for computing independent tasks in the cloud. *Proceedings of 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, 2010; 327–334.
27. Weiwei L, Yongjuen Q, Zhenyu W, *et al*. Independent tasks scheduling on tree-based grid computing platforms. *Journal of Software* 2006; **17**(11): 2352–2361.
28. Bharadwaj V, Robertazzi TG, Ghose D. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press: Los Alamitos, CA, 1996.
29. Hong B, Prasanna VK. Adaptive allocation of independent tasks to maximize throughput. *IEEE Transactions on Parallel and Distributed Systems* 2007; **18**: 1420–1435.