International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:5, No:3, 2011

Virtual Machines Cooperation for Impatient Jobs under Cloud Paradigm

Nawfal A. Mehdi, Ali Mamat, Hamidah Ibrahim, Shamala K. Syrmabn

Abstract—The increase on the demand of IT resources diverts the enterprises to use the cloud as a cheap and scalable solution. Cloud computing promises achieved by using the virtual machine as a basic unite of computation. However, the virtual machine pre-defined settings might be not enough to handle jobs QoS requirements. This paper addresses the problem of mapping jobs have critical start deadlines to virtual machines that have predefined specifications. These virtual machines hosted by physical machines and shared a fixed amount of bandwidth. This paper proposed an algorithm that uses the idle virtual machines bandwidth to increase the quote of other virtual machines nominated as executors to urgent jobs. An algorithm with empirical study have been given to evaluate the impact of the proposed model on impatient jobs. The results show the importance of dynamic bandwidth allocation in virtualized environment and its affect on throughput metric.

Keywords—Insufficient bandwidth, virtual machine, cloud provider, impatient jobs.

I. Introduction

MANY enterprises, organizations and governmental departments are responsible for time critical jobs and these jobs need to be served quickly. In addition to above, these agencies face IT problems because of the huge growth in applications, data and solution sizes.

Dave Murphy[1], senior vice president at performance testing vendor says *"The cloud allows for large amounts of computing power over short periods of time - like during a disaster - so government agencies can respond to anything in the world"*. He also noted *"The government could utilize the cloud compute power on an as-needed basis. All of a sudden - like with the oil spill off the coast of Louisiana - they would need to bring resources to bear, such as money, people and equipment. Part of the equipment is the computing power"*. This kind of urgent jobs where the time plays an essential role in their execution are called impatient jobs.

Impatient jobs are the jobs that need resources as soon as they enter the system and to be executed as soon as possible [2]. More recently, these jobs have a considerable amount of data to be staged in before execution and staged out after. Each job could have two deadlines in its QoS namely: start and finish deadlines, which are described in more details at section III.

Many experts proposed that cloud computing is a solution to these problems such that each agency can execute its jobs via the cloud and expand their requirements based on the situation.

The nearest cloud computing definition to our work is given by [3]:*Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs*. From the above definition, cloud computing is based on virtualization [4], [5] in its infrastructure which can be described as an emerging IT paradigm that separates computing functions and technology implementations from physical hardware. One of the main reasons that clouds adopt virtualization is the configurability of Virtual Machine (VM) [6].

In cloud computing or more precisely in Infrastructure as a Service (IaaS), each cloud provider has a set of predefined VMs images and the customer should select one of them for his job(s). However, the predefine bandwidth of the nominated VM might be not enough to meet the job start deadline, if it has huge data to be staged in before execution.

Several VMs can be installed on a physical machine and share its hardware components. These components can be CPU, memory, disk storage, or network bandwidth. It can be looked like a set of independent systems that sharing the same hardware [7]. While each physical machine has a fixed amount of bandwidth, each VM should have a share of that bandwidth to be used for its stage-in and stage-out processes. However, some VMs might have idle bandwidth if they are busy running their application. The idle bandwidth can be used while there is an ability to reconfigure the VMs setting. Some studies have shown that each VM may consume I/O bandwidth in unpredictable way [8], [9].

While IaaS builds of the idea of virtualization [10] or VMs images [11], and there is an ability to change or reconfigure the VM settings, we can use the idle bandwidth amount to speed up the stage-in process for the jobs that have critical start-deadlines or what we can called them *impatient jobs*.

This paper tackles the problem of *insufficient bandwidth amount specified to a particular VM nominated as the best executor for impatient job from data location point of view.* This work proposed an algorithm that computes the necessary bandwidth required to meet the job start deadline. The proposed algorithm maps the jobs with inspiration by Minimum Completion Time scheduling algorithm (MCT) under cloud environment. Problem and its formulations are described in more details in section III.

The main contributions of this work are:

- Divide the VM execution into periods and formulate the boundaries of these periods (Section III).

N. Mehdi, A. Mamat, H. Ibrahim, and S. symban are in the Faculty of Computer Science and Information Technology, University Putra Malaysia, Serdang, Selangor, 43400, Malaysia

- Propose an algorithm that tries to find a donor VM, which is busy with execution its application(i.e. has finished stage-in process) to have some of its resources without overlapping (Section IV).
- Transplant the proposed algorithm in an adopted immediate mode scheduling algorithm and test both algorithms using cloudsim simulator (Section IV and V).

The reset of this paper is organized as follows: section II gives the most related work are presented. Section III forms the problem in mathematical way and lists its objectives. Section IV illustrates the system model and the pseudo code of the proposed algorithm. Experiments and results are shown in section V while the section VI gives the conclusion and future work.

II. Related Work

Foster et al. in his paper [12] proposed a model for predicting time overheads involved in using the VMs. In their work, they use the suspend/resume/migrate capability of VMs to support advance reservation. They showed that, using the suspend/resume/migrate capability of VMs allowed the advance reservations to be supported efficiently, without the utilization problems commonly associated with them. They implemented their work in lease management software called Haizea that gives better results with this model. Our work tries to borrow resources (i.e. bandwidth) from running VMs (i.e. the VMs that are currently busy in their execution). Like Foster, we have to respect the donor VM by predicting the execution periods to return back the lended resources.

Kuriakose in his paper [9] presented a command-line tool for Network bandwidth differentiation in Xen which is used as a virtualization hypervisor [13] in many cloud provider like Amazon, GoGrid, Flexiscale, and RightScale. Their tool xmsetbw can dynamically reconfigure the I/O bandwidth allocated to a VM to enhance the existing scheduler to offer weighted fair scheduling requested across VMs. Their experimental evaluation verified the correctness of their tool when I/O bandwidth is changed dynamically.

This work supports our idea by providing the needed tool to implement it in real system. A classic VM enables multiple independent and isolated operating systems to run on one physical machine, efficiently multiplexing system resources of the host machine [14]. It provides a secure and isolated environment for application execution [15]. Compared with a physical machine, it is highly customizable in terms of hardware (such as CPU, memory, and disk space) and software (such as operating system and applications) and can be easily check pointed and migrated to achieve host load balancing [16], [17], [18], [19]. Recent technical advances in VM development have made adaptive hardware configuration possible. For example, processors and/or memory can be dynamically added to or removed from a running system [20], [5]. All these characteristics make VMs highly manageable resource containers for applications in utility environments.

Immediate mode job scheduling is the mode of scheduling the jobs as soon as it received by the system. In other words, it is the scheduling of impatient jobs or the scheduling of VIP jobs that have deadline for their executions. Hak Du Kim and Jin Suk Kim [21] took the problem of scheduling independent jobs in Grid environment. They proposed an online scheduling technique and compared it with the other techniques using simulations. In their research, they did not take the input/output files in considerations. They based on job length (i.e. execution time). Fatos Xhafa et al. [2] considered the problem of allocation jobs using immediate mode in grid environment. They implemented five scheduling methods and used four parameters to measure the performance of the system; namely, 1) makespan, 2) flow time, 3) resource utilization and 4) matching proximity. In this paper, they did not take the data-location in consideration of the scheduling process. They based on the job execution time.

S. Ghanbari and M. R. Meybodi [22] worked on online scheduling techniques in computation grids. They proposed four algorithms for scheduling jobs based on learning automata.They did a simulation to evaluate the performance of their system. The four algorithms use the job length as a factor for Comparison.

III. Problem and Model Formulation

We refer to [23], [24], [25], [26] for the proposed system topology.

Let cloud provider $\mathcal{P} = \{\mathcal{D}, \mathcal{V}\}$ consists of set of datacenters $\mathcal{D}$ and set of VM images $\mathcal{V}$. Virtual machine v_q is described by set of specifications $< CPU, RAM, BW, ... >$. Each VM v_q is hosted by Physical Machine (PM) p_k which is in turn represents the basic hardware unit of computation in the datacenter. Let $\mathcal{B}_{p_k}$ is the PM p_k bandwidth and $\mathcal{B}_{v_q}$ denotes the VM bandwidth quota, such that $\mathcal{B}_{p_k} \geq \mathcal{B}_{v_q}$ if VM v_q is hosted by PM p_k. The time needed to reconfigure the bandwidth quota for VM v_q is denoted by μ.

Let $\mathcal{J}$ is the set of independent dynamic arriving jobs. Each job j_e is described by a set of specifications, such that: $j_e = \{\mathcal{L}_e, at_e, sd_e, fd_e, Fin_e, Fout_e\}$, where $\mathcal{L}_e$ is the job length in MIPS, at_e is the job arrival time, sd_e is the job start deadline, fd_e is the job finish deadline, Fin_e is the set of input files, and $Fout_e$ is the set of output files. Job deadlines have two hard constrains and should be respected by the scheduler. Formally 1) $sd_e \geq \alpha_e$ which means job execution is useless if the job starting time α_e is after the deadline sd_e; 2) $fd_e \geq \delta_e$, which ensures to complete the job before its finish deadline fd_e where δ_e is the job finish time. Now, let j_e has input size equal to $\mathcal{I}_e$ and can be assigned to a beneficiary VM v_b that has bandwidth equal to $\mathcal{B}_{v_b}$. Stage-in proces s depends totally on the bandwidth and VM storage size. We assume that there is an enough storage for each VM. It is possible to speed up the stage-in process by increasing the amount of bandwidth for the nominated VM.

Let v_d denotes the donor VM that has bandwidth equal to $\mathcal{B}_{v_d}$ and computation capacity equal to $\mathcal{S}_{v_d}$. The required bandwidth for stage-in process for job j_e is computed by 1 while the extra bandwidth amount required for v_b is computed by 2 as shown below.

$$\mathcal{B}_{total} = \frac{\mathcal{I}_e}{sd_e - CT} \quad (1)$$

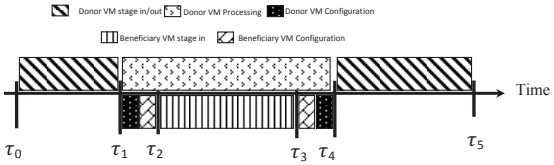

Fig. 1. Job execution time line

$$\mathcal{B}_{need} = \mathcal{B}_{total} - \mathcal{B}_{v_b} \quad (2)$$

where $\mathcal{B}_{total}$ is the best bandwidth amount that can finish the stage-in process before the job's start deadline. If $\mathcal{B}_{total} \leq 0$ then it means the time is over and the start deadline can not be met. CT is the current system time and $\mathcal{B}_{v_b}$ is the predefined bandwidth quota for VM v_b.

The proposed algorithm finds the best VM to borrow some bandwidth from it for a while. Finding an idle bandwidth period for the donor VM is the challenge problem. Figure 1 depicts the proposed time line of job execution under VM with reconfiguration in its bandwidth size. It divides the execution process to periods each one has a start time as can be seen from 3 to 8.

$$\tau_0 = v_d \text{ starting time} \quad (3)$$

$$\tau_1 = \tau_0 + \sum_{f \in Fin_d} \frac{Size(f)}{min(\mathcal{B}_{v_d}, \mathcal{B}_f^{src})} \quad (4)$$

$$\tau_2 = \tau_1 + 2 * \mu \quad (5)$$

$$\tau_3 = \tau_2 + \sum_{f \in Fin_b} \frac{Size(f)}{min(\mathcal{B}_{v_b} + \mathcal{B}_{v_d}, \mathcal{B}_f^{src})} \quad (6)$$

$$\tau_4 = \tau_1 + \frac{\mathcal{L}_d}{\mathcal{S}_{v_d}} \quad (7)$$

$$\tau_5 = \tau_4 + \sum_{f \in Fout_i} \frac{Size(f)}{min(\mathcal{B}_{v_d}, \mathcal{B}_f^{dis})} \quad (8)$$

The start time is denoted by τ_0, which indicates the time of starting the stage-in phase for the donor VM. The stage-in phase finishs at time indicated by τ_1 that is equal to τ_0 plus the time needed to fetch all the input files as can be seen in 4, where Fin_d denotes the list of input files attached with the job under the VM v_d and $\mathcal{B}_f^{src}$ is the bandwidth value of the sites where the input files are located. This time is also the start time for reconfiguring the bandwidth size process. As aforementioned, μ is the time needed to reconfigure the donor VM by decreasing its bandwidth size or beneficiary VM by increasing its bandwidth size. The finish time of this process can be seen in 5 that is denoted by τ_2. The time needed to fetch all the input files for the impatient job, which is denoted by Fin_b is computed in 6. After fetching all the input files for the beneficiary VM, the bandwidth sizes should be returned back to the donor VM. τ_3 gives the start time of reconfiguring the bandwidth sizes. At time τ_4, which is computed using 7 everything should return back to its normal. In this time the donor VM v_d finishs execution the job that has length $\mathcal{L}_d$. The Last time is denoted by τ_5 which is the finish time for the donor VM as can be seen in 8.

The time table of each job is save in a form of tuple $r =< v_d, p_j, \tau_0, \tau_1, \tau_4, \tau_5 >$ in a special data structure called Time Line Registry table (TLR), which is used later in scheduling algorithm (section 4.2).

The proposed algorithm generates three events in three particular times, which are:

1) **Lending Event**:This event starts working after time τ_1 and responsible for lending bandwidth from v_d and pass it to v_b.
2) **Triggering event**:This event starts working at time τ_2 and gives the trigger to v_b VM to start its stage-in process.
3) **Roll back event**:This event starts working at time τ_3 and returns the original configurations to its normal (i.e. to the state of configurations before time τ_1).

The objectives of this algorithm can be described as:

- **Maximize Virtual Machine Bandwidth Utilization:** This work tries to improve the bandwidth share for each VM such that it uses the idle period if the VM busy executing. Bandwidth utilization formula is given in 9.

$$\mathcal{U}_v = \frac{\mathcal{T}_v \times 100}{\mathcal{B}_v \times \psi_v} \quad (9)$$

  where $\mathcal{U}_v$ is the bandwidth utilization for VM v, and $\mathcal{B}_v$ is the bandwidth quota, ψ_v is the duration which is the VM life time and it is equal to the VM release time to the VM creation time, and $\mathcal{T}_v$ is the amount of data transferred during the VM life.
- **Maximize Throughput:** The number of successful job execution is called *throughput*. It is computed using 10.

$$Throughput = \sum_{j \in \mathcal{J}} \mathcal{X}_j \quad (10)$$

  where $\mathcal{X}_j$ is a condition veritable indicates the execution if the job as shown in 11.

$$\mathcal{X}_j = \begin{cases} 1, & \text{job } j \text{ has finished execution} \\ 0, & \text{otherwise} \end{cases} \quad (11)$$

- **Minimize Job Failure ratio:** The main objective of this work is to provide enough resources to meet the jobs requirements. This objective computes the number of failed jobs as shown in equation 12.

$$\mathcal{F} = \sum_{j \in \mathcal{J}} 1 - \mathcal{X}_j \quad (12)$$

Several constraints have been taken in account with this work. These are:

1) $\mathcal{B}_{p_k} \geq \sum \mathcal{B}_{v_q} \ \forall v_q \in p_k$
2) $\tau_0 \leq \tau_1 < \tau_2 = \tau_1 + \mu \leq \tau_3 < \tau_4 = \tau_3 + \mu \leq \tau_5$

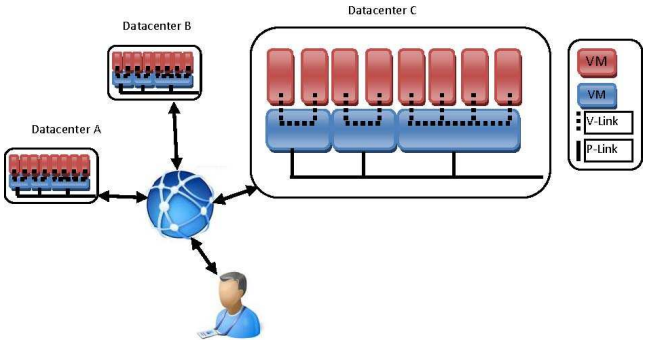

Fig. 2. System model

3) $\frac{I_e}{\mathcal{B}_{total}} + CT + \delta \leq sd_e$

Bandwidth sharing between VMs is formulated in the first constraint, which means the summation of allocated bandwidth to all VMs that are hosted by PM p_k should be not more than the PM bandwidth. The second constraint ensures the sequence of events while the third constraint ensures the lent bandwidth is enough to meet the start deadline.

IV. System Model

Cloud providers use datacenters as an infrastructure to serve their customers. The proposed model consists of set of datacenters connected to the internet. Each datacenter has set of physical machines that are used to host the virtual machines. Each set of virtual machines share the amount of bandwidth of the host physical machine.

The user sends his/her urgent job through cloud broker to the cloud exchange. The cloud exchange has the list of cloud providers and their offers and current states that is updated by the cloud coordinators. The cloud exchange maps the request to the best cloud provider that can meet the client QoS requirements.

The proposed algorithm tries to find a donor VM to take some bandwidth from its share and adds it to the needed VM. It registers the VM time line that is described earlier in a registry to be used later by the scheduler. This registry is updated with each VM creation and leasing.

With each arrival job the algorithm finds the amount of bandwidth that can staged in its input files within the deadline. Then it checks the available VMs images to find the suitable image. If the current VM image has bandwidth amount not enough to meet the start deadline for the current job, it tries to lend bandwidth from the available running VMs.

Algorithm 1 illustrates the main steps for the proposed algorithm. This algorithm at step 1 scans all the received jobs and tries to map them to resources. Step 2 computes the total bandwidth needed for the job j to meet its starting constraint. Step 3 collects the available free (i.e. unused) bandwidth and named it φ. A temporary value $minvalue$ is used to find the minimum execution time offered from all available virtual machines. The *for* loop statement at step 5 scans all the available VM. Initially it checks if this VM v has an enough bandwidth for the job j; if so, then go to step 17 and computes

Algorithm 1: Cloud Coordinator Strategy

1 **foreach** $j \in \mathcal{J}$ **do**
2 $\quad \mathcal{B}_{total} \leftarrow \frac{\mathcal{I}_e}{sd_e - CT - \delta}$;
3 $\quad \varphi \leftarrow FreeBW(\mathcal{D})$;
4 $\quad minvalue \leftarrow +\infty$;
5 $\quad$ **foreach** $v \in \mathcal{V}$ **do**
6 $\quad\quad \mathcal{B}_{added} \leftarrow \mathcal{B}_v$;
7 $\quad\quad$ **if** $\mathcal{B}_v < \mathcal{B}_{total}$ **then**
8 $\quad\quad\quad \mathcal{B}_{added} = 0$;
9 $\quad\quad\quad$ **if** $\mathcal{B}_v + \varphi < \mathcal{B}_{total}$ **then**
10 $\quad\quad\quad\quad$ **foreach** $e \in TLR$ **do**
11 $\quad\quad\quad\quad\quad$ **if** $(e.\tau_1 < CT) and (e.\tau_4 > j.\tau_3) and (\mathcal{B}_v + e.\mathcal{B}_v \geq \mathcal{B}_{total})$ **then**
12 $\quad\quad\quad\quad\quad\quad$ Select e as a donor VM;
13 $\quad\quad\quad\quad\quad\quad \mathcal{B}_{added} \leftarrow \mathcal{B}_v + e.\mathcal{B}_v$;
14 $\quad\quad\quad\quad\quad\quad$ Break;
15 $\quad\quad\quad$ **else**
16 $\quad\quad\quad\quad \mathcal{B}_{added} \leftarrow \mathcal{B}_v + \varphi$;
17 $\quad\quad$ **if** $\mathcal{B}_{added} > 0$ **then**
18 $\quad\quad\quad \chi^{in}_{j,v} \leftarrow \sum_{f \in Fin_t} \frac{Size(f)}{min(\mathcal{B}_{added}, \mathcal{B}_f)}$;
19 $\quad\quad\quad \chi^{out}_{j,v} \leftarrow \sum_{f \in Fout_t} \frac{Size(f)}{min(\mathcal{B}_{added}, \mathcal{B}_f)}$;
20 $\quad\quad\quad P_{j,v} \leftarrow \frac{\mathcal{L}_j}{\mathcal{S}_{v_d}}$;
21 $\quad\quad\quad minvalue, \hat{v} \leftarrow min\{minvalue, \Re + \chi^{out}_{j,v} + \chi^{in}_{j,v} + \chi^{code}_{j_e,V_k} + P_{j,v}\}$;
22 $\quad$ **if** $minvalue \neq \infty$ **then**
23 $\quad\quad$ Map j to $\hat{v}$;
24 $\quad\quad$ *Generate the lending event at time* CT;
25 $\quad\quad$ *Generate the trigger event at time* $CT + 2 \times \mu$;
26 $\quad\quad$ *Generate the roll back event at time* $j.\tau_3$;

the estimated time to complete. Otherwise, checks if adding the free bandwidth is enough (step 9); if not then checks all the running VMs that have entries in the time line registry TLR. If the available entry (i.e. VM) is suitable to be a donor VM, then nominate it. Steps 18-21 computes the job completion time while step 23 maps the job to the nominated VM if there is an enough bandwidth for it. Step 24 starts the lending event at time CT, which is equal to the job arrival time or τ_1 as shown in 4. The stage-in process for job j is triggered after $2 \times \mu$ from the CT as shown in step 25. Last events is the rollback event, which is starts at time $j.\tau_3$ and explained formally in 6.

This algorithm calculates the completion time for each job on each VM image and assigns the job to the VM that gives the earliest completion time. This algorithm uses (13) to estimate the total time needed to finish executing a job on a selected virtual machine.

$$FT_{j_e,V_k} = \chi^{in}_{j_e,V_k} + \chi^{code}_{j_e,V_k} + \Re + P_{j_e,V_k} + \chi^{out}_{j_e,V_k} \quad (13)$$

where $\chi^{in}_{j,v}$ represents the time needed to fetch all the necessary input files for the job j from the sources to the nominated VM v. $\chi^{code}_{j,v}$ denotes the time needed to fetch the application

code to the nominated VM v. Each virtual machine needs a particular time for installing, booting, and loading. This time is represented in $\Re$. Job execution time, which is equal to job length divided by VM speed is represented by $P_{j,v}$. $\chi_{j,v}^{out}$ represents the time needed to send all the output files for the job j to their destinations.

This algorithm has been improved to keep pace with the significant growth in jobs data by taking the application, input and output files locations in account. VM ready time is added to total completion time to fit the characteristics of intercloud paradigm.

V. Performance Evaluation

In the absence of real traces from real cloud providers, we generate the input workload randomly. Casazza et al. [27] present a workload methodology to characterize the performance of servers exploiting virtualization technologies to consolidate multiple physical servers. They combine the workload of web server, an email server, and a database application to reflect the variation of application that can be run on cloud computing. We generate synthetic data distributions workload as we now describe.

As described in section III, we assume a certain number of users, jobs, and cloud providers. Jobs workload are selected randomly [28] with uniform distribution between 500 MB to 2000 MB. Each job has set of input and output files selected randomly between 1 and 6. Job length is a function to the total input size for 300D if we assume D is the total input size. Simulation is the process of imitation of the real system. Because of the difficulty in testing the proposed system in a real system, a simulation evaluation has been conducted on four datasets. CloudSim [29] is a discrete event simulator that is used to simulate cloud environments. Cloudsim has the ability to create data centers, virtual machines and physical machines and configure system brokers, system storage, etc. The proposed algorithm has been tested and evaluated on the two real datasets.

Table I specifies the simulation parameters used for our study. Two performance metrics have been used to evaluate the proposed model. Virtual machine bandwidth utilization is computed using 9. It computes the percentage of utilizing VM bandwidth.

TABLE I
CloudSim Configurations

Item	Value
Number of data centres	14
Number of VM	100
Number of CPU/VM	1
CPU Speed/VM	1Ghz, 2GHz, 2.5GHz, 3GHz
Number of jobs	100,200,300,400,500

Several experiments have been done to test the impact of the proposed system on executing impatient jobs under cloud paradigm.

The first set of experiments using the cloudsim simulator is done to evaluate the impact of the proposed algorithm on bandwidth utilization.

Fig. 3 depicts Virtual Machine Average bandwidth Utilization (VMAU) with bandwidth lending and without. The variety in datasets sizes allows the experiments to evaluate the system with multiple loads. This figure depicts the bandwidth utilization for the VMs that finish execution their jobs or in another words for the jobs that meet their requirements. As discussed earlier, the job size is a function to the size input files and while we assumed the current jobs are huge in data size, we can see the utilization is low. This is because of the most of the time is spent in execution while the bandwidth is idle. The same reason can explain the small difference in bandwidth utilization when the lending algorithm is active. In the same figure, there are some experiments have very small difference in bandwidth utilization for example in dataset with job number 5, 70, 200. The reason is the random generation for the jobs does not give many job in need for high bandwidth.

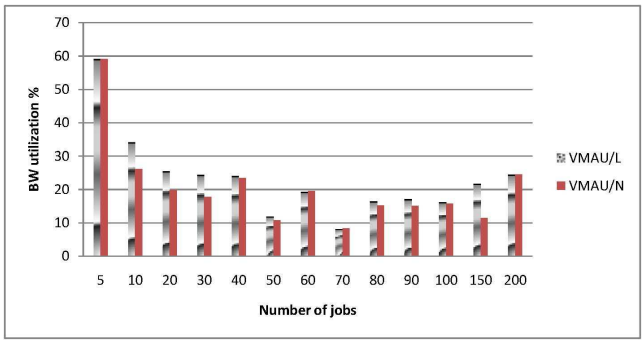

Fig. 3. Virtual Machine Average Bandwidth Utilization

Fig. 4 depicts the number of jobs that failed in finding proper VM. The generation of datasets are done to have a ratio of urgent jobs. This figure depicts the same number of experiments that are done on the previous figure. It is quite clear to see the difference in number of failed jobs. In the ordinary mapping system (i.e. the mapping without bandwidth lending) many jobs fail to find the proper VM that can meet the requirement. The proposed algorithm checks the start deadline and then the finish deadline of the each job by estimating the data transfer time and the job execution time. For the failed jobs, either they do not meet the start deadline or the finish deadline.

From the figure, its quite clear to see that the number of failed jobs increases as the number of jobs in each set increases. This is because of increase the number of jobs that need fast attention.

In the same figure, it is possible to see some failure even while the bandwidth lending algorithm is active. This is because of a small limitation in the proposed algorithm. The algorithm needs some VM in running state to borrow or lend bandwidth from them. So, in the random generation, if the impatient jobs came in the beginning which means they have the nearest arrival time. The algorithm cannot find a donor VM to lend bandwidth from it.

Fig. 5 shows the throughput of each dataset. Actually, the throughput is the reflection of number of job failure. It is computed using equation 10. In this figure, its clear to

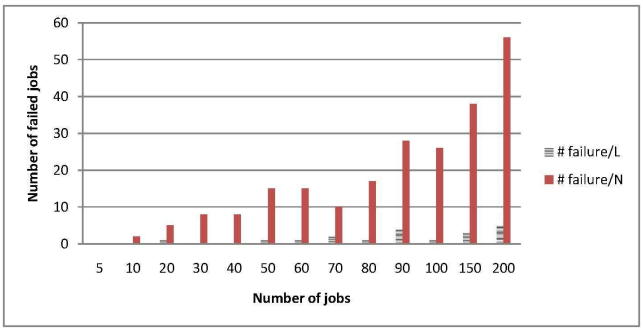

Fig. 4. Number of failed jobs

see the improvement in the number of job execution. As aforementioned in the previous paragraph, there is some failure even in the lending algorithm and again this is because of the random generation which put the impatient jobs in the beginning.

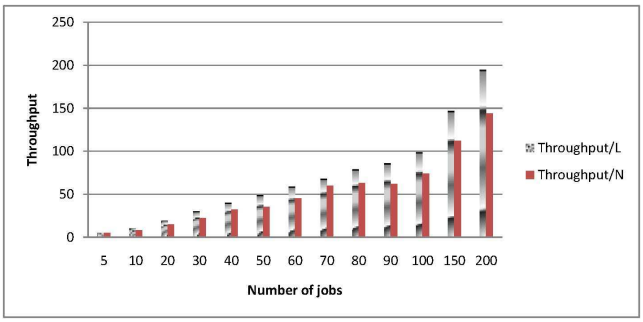

Fig. 5. Total System Throughput

VI. Conclusion

In this paper, a bandwidth lending algorithm is proposed. The algorithm takes the impatient jobs as its scope and tries to allocate more bandwidth than the predefine size for each VM to speed up the stage-in process which implies to meet their QoS requirements. This algorithm divides the job execution time into periods and tries to predict the best VM that can donor its bandwidth while it is busy with execution. Simulation is done on two real datasets to test the performance of the proposed algorithm that is compared with adopted algorithm to cloud environment. The results show that BWLMCT can provide better system performance if the jobs have restrict and hard deadlines.

Future work is going on to improve the proposed algorithm by allowing it to check more than one virtual machine simultaneously. Current pricing methods are not fair with this method for both sides: the donor and the recipient.

References

[1] M. J. Kronfeld, "Experts believe cloud computing will enhance disaster managemen," *GSN: Government Security News*, pp. 34–39, 2010.

[2] F. Xhafa and A. Leonard, "Immediate mode scheduling of independent jobs in computational grids," in *21st International Conference on Advanced Information Networking and Applications, 2007. AINA'07*, 2007, pp. 970–977.

[3] L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[4] M. Kesavan, A. Ranadive, A. Gavrilovska, and K. Schwan, "Active CoordinaTion (ACT)-toward effectively managing virtualized multicore clouds," in *Cluster Computing, 2008 IEEE International Conference on*. IEEE, 2008, pp. 23–32.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM, 2003, p. 177.

[6] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee, 2008, pp. 1–10.

[7] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum, "Disco: Running commodity operating systems on scalable multiprocessors," *ACM Transactions on Computer Systems (TOCS)*, vol. 15, no. 4, pp. 412–447, 1997.

[8] D. Ongaro, A. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2008, pp. 1–10.

[9] K. Mathew, P. Kulkarni, and V. Apte, "Network bandwidth configuration tool for xen virtual machines," in *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*, jan. 2010, pp. 1 –2.

[10] V. Manetti, P. Di Gennaro, R. Bifulco, R. Canonico, and G. Ventre, "Dynamic virtual cluster reconfiguration for efficient iaas provisioning," in *Proceedings of the 2009 international conference on Parallel processing*, ser. Euro-Par'09. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 424–433. [Online]. Available: http://portal.acm.org/citation.cfm?id=1884795.1884844

[11] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, "Scientific cloud computing: Early definition and experience," *High Performance Computing and Communications, 10th IEEE International Conference on*, vol. 0, pp. 825–830, 2008.

[12] B. Sotomayor, R. Montero, I. Llorente, and I. Foster, "Resource leasing and the art of suspending virtual machines," in *2009 11th IEEE International Conference on High Performance Computing and Communications*. IEEE, 2009, pp. 59–68.

[13] B. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *2009 Fifth International Joint Conference on INC, IMS and IDC*. IEEE, 2009, pp. 44–51.

[14] R. Goldberg, "Survey of virtual machine research," *IEEE Computer*, vol. 7, no. 6, pp. 34–45, 1974.

[15] R. Figueiredo, P. Dinda, and J. Fortes, "A case for grid computing on virtual machines," in *23rd International Conference on Distributed Computing Systems, 2003. Proceedings*, 2003, pp. 550–559.

[16] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, p. 286.

[17] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 377–390, 2002.

[18] M. Kozuch and M. Satyanarayanan, "Internet suspend/resume," 2002.

[19] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. Networked Systems Design and Implementation*, 2007.

[20] S. Pinter, Y. Aridor, S. Shultz, and S. Guenender, "Improving machine virtualisation with'hotplug memory'," *International Journal of High Performance Computing and Networking*, vol. 5, no. 4, pp. 241–250, 2008.

[21] H. Kim and J. Kim, "An online scheduling algorithm for grid computing systems," *Grid and Cooperative Computing*, pp. 34–39, 2004.

[22] S. Ghanbari and M. Meybodi, "On-line mapping algorithms in highly heterogeneous computational grids: A learning automata approach," in *International Conference on Information and Knowledge Technology (IKT05)*, 2005.

[23] L. Mei, W. Chan, and T. Tse, "A tale of clouds: Paradigm comparisons and some thoughts on research issues," in *2008 IEEE Asia-Pacific Services Computing Conference*. IEEE, 2008, pp. 464–469.

[24] D. Bernstein and D. Vij, "Using XMPP as a transport in Intercloud Protocols." submitted to 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'10), for publication June, 2010.

[25] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the Intercloud-Protocols and Formats for Cloud Computing Interoperability," in *Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services-Volume 00*. IEEE Computer Society, 2009, pp. 328–336.

[26] R. Buyya, R. Ranjan, and R. Calheiros, "InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services," *Algorithms and Architectures for Parallel Processing*, pp. 13–31, 2010.

[27] J. Casazza, M. Greenfield, and K. Shi, "Redefining server performance characterization for virtualization benchmarking," *Intel Technology Journal*, vol. 10, no. 3, pp. 243–251, 2006.

[28] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *11th IEEE International Symposium on High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings*, 2002, pp. 352–358.

[29] R. Calheiros, R. Ranjan, C. De Rose, and R. Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," *Arxiv preprint arXiv:0903.2525*, 2009.