Advantages, Challenges and Optimizations of Virtual Machine Scheduling in Cloud Computing Environments

Hadi Salimi, Mahsa Najafzadeh, and Mohsen Sharifi

***Abstract*—Cloud computing enjoys the many attractive attributes of virtualization technology, such as consolidation, isolation, migration and suspend/resume support. In this model of computing, some desirable features such as scalability are provided by means of a new type of building blocks called virtual machines (VMs). As with any other construction block, VMs have their own scheduling challenges and advantages. This paper presents the major differences between scheduling VMs and other schedulable blocks such as processes and explains why traditional scheduling techniques used in operating systems to schedule processes or threads are not suited to VMs. New techniques to help co-scheduling virtual resources in a concurrent environment are proposed and simulated on an extension of CloudSim simulator. Simulation results of virtual processor co-scheduling show comparatively higher system performance than the methods that do not use co-scheduling.**

***Index Terms*—Cloud computing, co-scheduling, virtualization.**

I. Introduction

Cloud computing is an emerging paradigm that aims to share various resources and distribute services transparently among massive users of a computer network. It could be considered as a pool of virtualized computer resources with dynamic composition and deployment of software services. Some researchers [1] consider Clouds as a complement of Grid system environments that provide more efficient resource management. In particular, they believe Clouds allow the dynamic scaling of applications by provisioning of resources via virtualization. In addition, by monitoring virtualized resources, it would be possible to support dynamic load balancing and re-allocations of resources.

To support the mentioned features, virtual machines (VMs) have been proposed as the primitive building blocks of a Cloud environment. Using these blocks, VMs can be easily distributed to different physical machines or consolidated to the same machine in order to balance the load or simply increase the utilization. In addition, by using the intrinsic properties of a VM, it would be possible to move a running application and all its dependents to another machine without stopping its processes.

Scheduling the basic processing units on a computing environment has always been an important issue. As an example, the problem of scheduling processes or threads in operating systems has been studied for decades. On the other hand, scheduling tasks in a Grid environment has also been studied deeply in literature. As with the mentioned computing environments and the fact that VMs are more abstract than processes and tasks, the scheduling of VMs on a distributed Cloud infrastructure requires further studies.

In this paper we present the basic advantages and challenges of scheduling VMs compared to other scheduling methods applied to processes or tasks used in traditional computing environments. As an optimization for VM scheduling, we introduce virtual CPU (VCPU) co-scheduling (Gang scheduling) [2]. Analogous to the basic co-scheduling algorithm that schedules related processes to run on different processors at the same time, our method tries to map related VCPUs to real processors simultaneously. Using this technique, any pair of related processes that run on VMs run faster leading to an overall higher performance. This scheduling method avoids unnecessary VM blocks. We have evaluated this technique by implementing it on an extended version of CloudSim [3,4] simulator. We changed the simulator in such a way to be able to model task dependencies and synchronization points.

The remainder of this paper is organized as follows. Section 2 presents scheduling concerns of VMs and the major differences between scheduling VMs and other processing units. Section 3 presents the benefits of scheduling a set of VMs on a computer environment. Sections 4 and 5 present our proposed optimization technique and its evaluation and finally Section 6 concludes the paper.

II. VM Scheduling Challegnges

At the first glance, scheduling a number of VMs looks similar to scheduling other processing units such as processes or threads. However, having a closer look at this problem, it appears that VM scheduling is more troublesome. In the remaining three sub-sections, we explain some of these challenges.

A. Two-Level Scheduling

In contrast to scheduling of processes or threads wherein executing units are mapped directly to physical resources in one level, on a virtualized environment resources need to be scheduled in two levels as it is depicted in Fig. 1. In the first level, the operating system scheduler maps running execution units into virtual resources provided by VMM. In the second level, the VMM scheduler maps virtual resources presented to guest operating systems into real hardware. Given that most first-level schedulers are implemented by commodity operating systems and the second level schedulers are usually unaware of the scheduling policies used by these operating systems, this semantic gap can well lead to inefficient

Manuscript received February 19, 2012; revised March 31, 2012.

Authors are with the School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran. (e-mail:hsalimi@iust.ac.ir, mahsa_najafzadeh@comp.iust.ac.ir, msharifi@iust.ac.ir).

scheduling of resource.

B. Higher Level Abstraction

In an operating system that maps a set of processes into a set of resources, details of system processes are known to the scheduler. For example, the scheduler can identify related processes that need to be synchronized at specific execution points at run time by tracing process communication patterns. This kind of knowledge can help the operating system to use a time-space shared schedule to execute dependant processes more efficiently.

On the other hand, regarding this fact that VMs provide a higher level of abstraction to their management environment, i.e. VMM, obtaining such detailed knowledge about their internals is impossible. Hence, the VMM cannot determine any correlations between tasks that are running on the VMs. This lack of knowledge that is generally attributed to the high level of abstraction of VMs, may well lead to improper resource mapping of virtual resources to real resources.

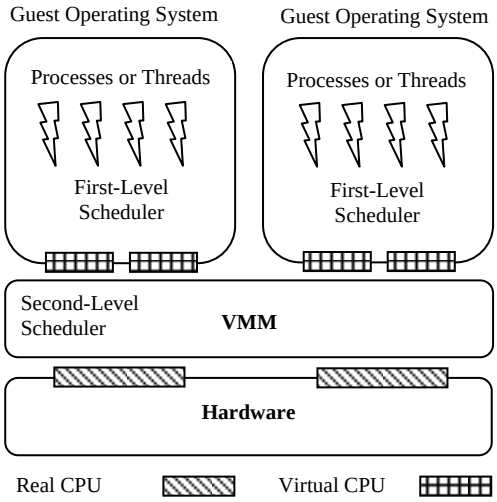

Fig. 1. Two level CPU scheduling in a virtualized environment.

C. Unpredictable Behavior

The right prediction of the behavior of executing units in a computer system can be used to schedule resources more efficiently [5,6]; prediction allows the scheduler to allocate resources based on the estimated required resources. For example, if the scheduler can predict that the jobs running on a remote machine in a distributed system are about to be completed, it can rightly dispatch the next ready job to that node. As another example, predicting the execution time of a job helps to select the best job that backfills the next one.

In contrast to a sequential process or thread that executes a single stream of instructions and thus has a relatively predictable behavior, VMs act as a container for processes and threads. Therefore, predicting the behavior of a VM by just inspecting the instructions issued on its virtual processors is very hard, if not impossible.

III. VM Scheduling Advantages

Despite many difficulties that accompany the VM scheduling problem, using them as the basic building blocks of a distributed system is very beneficial. In this section, we discuss the two main benefits, namely, easy reservation and VM elasticity.

A. Ease of Resource Reservation

Since distributed systems are used in a widespread range of applications, the need for delivering special resources at particular times has been essential. As a result, a set of resource reservation mechanisms have been proposed [7,8]. The need for these techniques arises, when some parts of a heavy computational task require co-scheduling. For example, applications whose execution steps can be modeled as a workflow of independent tasks can be executed more efficiently by multilevel scheduling techniques.

One of the most important requirements of a resource reservation technique is task pre-emption. This feature is required because the scheduler may need to pause a task because of the start of a reservation contract on the same machine. Task pre-emption requires special services from operating system, such as checkpointing the task state frequently in order to resume it later. It is also possible to do the checkpointing operation at application level, by changing the code, which is not desirable.

To avoid changing the program code or using a special-purpose operating system, VMs come in handy. VMs can be easily checkpointed without requiring any changes to applications or operating systems. In addition, it is prudent to use them to provide a primitive for transparently vacating workloads to provide support for their migration.

B. VM Elasticity

Many scheduling challenges arise from the nature of processing units available in today's computer systems. As an example, due to the lack of flexibility of current processing blocks such as processes, many primitive requirements of a distributed scheduling such as migration hardly can be fulfilled. On the other hand, some inherent elasticity-related VM features such as live migration or consolidation can be regarded very beneficial to these schedulers.

Many scheduling systems try to balance the load on different nodes of a distributed system efficiently. Recent scheduling systems that are mostly based on load balancing [9], try to dispatch the jobs in a way to keep all system nodes busy. A major drawback of such a job scheduling is the lack of job pre-emption facilities. Due to the inherent properties of a job they cannot be easily suspended and resumed elsewhere. On the other hand, if jobs were able to be executed on an isolated environment like a VM, which could easily migrate between nodes, load balancing through job scheduling would be easier.

Other well-known scheduling algorithms could be augmented with elastic features of VMs too. As an example, many software solutions provided for distributed processing try to reduce the communication overhead of pairs of communicating entities. This is done either by statically determining the coupling degree of processing entities [10] and running them on the same machine, or by dynamically moving highly communicating ones to a single workstation.

By means of virtualization technology, augmented scheduling mechanisms can be implemented easier. Given the fact that the mentioned methods are based on dynamic

migration of executing modules, and that VMs play this role better than any other known alternative, a class of distributed schedulers could be promoted by these methods. VMs can be useful to other types of schedulers too that use different scheduling criteria such as power management [11].

IV. VM Scheduling Optimization

One of the main scheduling methods commonly used in concurrent systems is co-scheduling. This method tries to schedule related processing modules to be executed at the same time. If a concurrent application contains a set of processes that work closely together and if one of the executive ones tries to receive a message from those that are waiting for a processor share, this causes the running one to be blocked. On the other hand, ultimately other processes will be ready for execution but this time the situation is reversed and these processes should wait to interact with others. As a result, the application progresses more slowly compared to the case wherein all processes are independent.

The same mentioned case is valid for processes executed on one or two VMs. As a first example, consider the case in which four groups of concurrent processes are running on a VM as it is depicted in Fig. 2. Using the co-scheduling method embedded in the guest operating system's scheduler, each group of co-working processes are scheduled at the same time into two virtual processors in order to avoid extra process blocks. Suppose that at a given time, the first group comprising processes A and B is scheduled on two virtual processors and another group of processes comprising processes G and H are scheduled on the two remaining virtual processors. Since the VMM's scheduler that maps virtual processors to real ones is unaware of the load dependencies, it cannot properly decide on an efficient processor mapping. Hence, although the guest operating system is aware of the co-working processes, this knowledge cannot be used for the good of scheduling.

It should be noted that some VMs, like the ones defined on VMWare workstation [12], are not allowed to have more virtual processors than the number of real processors. But it should be taken into consideration that in a dynamic Cloud environment with many VMs migrating from one node to another, and given that the heterogeneity of underlying hardware, the case depicted in Fig. 2 is unavoidable.

As a second example, suppose that two co-working processes are executing on two different VMs. If the VMM knows about the dependency between these two processes, it can schedule the related VCPUs at the same time. Such a case may seem a bit strange, but with the use of advanced features of VMs such as cloning [13], there may well exist cases in which a child and a parent process execute on two different VMs. In this case, the co-scheduling of VCPUs onto which these two processes are mapped is beneficial.

It should be also noted that making the VMM aware of the policies used by the guest operating system's scheduler is possible in many ways. The first method could be defining a set of standard hypercalls in which the para-virtualized guest operating system can introduce its co-working processors to the VMM. This mechanism is straightforward, but it needs the hypercall interface to be changed. A simpler but less flexible method could be defining this information as a metadata to be used by the VMM to load VM images. Having provided such knowledge, the VMM can schedule VCPUs according to the mentioned default settings.

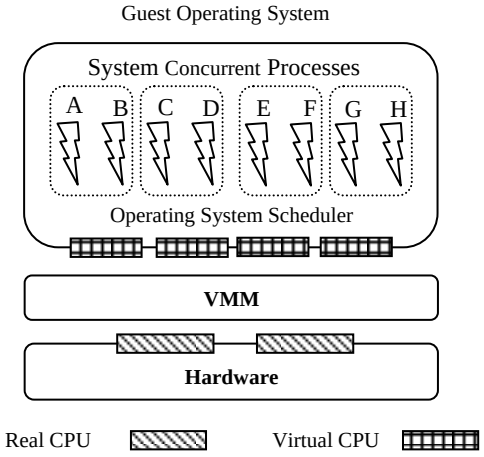

Fig. 2. Co-scheduling of concurrent processes on a VM.

We have extended the CloudSim simulator and run all the above discussed cases; the results and our detailed analysis of the results are reported in the next section. Let us however suffice to state here that the performance of highly concurrent jobs were increased under the proposed methods.

V. Performance Evaluation

In this section, we present the results of simulating two cases described in the previous section. In these two case studies, a job comprising four concurrent tasks and each task containing 800 million instructions was executed in a virtual environment. The hardware configuration in these experiments included two processors, each capable of executing 1000 million instructions per second (MIPS).

All simulations were performed on an extended version of CloudSim simulator. To measure the effect of task dependency on overall system performance, we changed parts of this simulator to be able to define co-working tasks and their dependency degree.

In the first experiment, we simulated a situation in which four tasks of a concurrent job were run on a single VM. Two of these tasks were co-working; hence needed to be synchronized on some points. For this case, the selected VM was equipped with four virtual processors and the underlying hardware had two real processors. The results of this experiment are depicted in Fig. 3.

As the results of this experiment show, the number of synchronization points changed from zero (no task dependency) to 200 points. When all tasks of the job were totally independent, co-scheduling did not affect system performance. On the other hand, when the number of synchronization points increased, the places in the code that must have waited for another task were increased. As a result, co-scheduling lead to better performance.

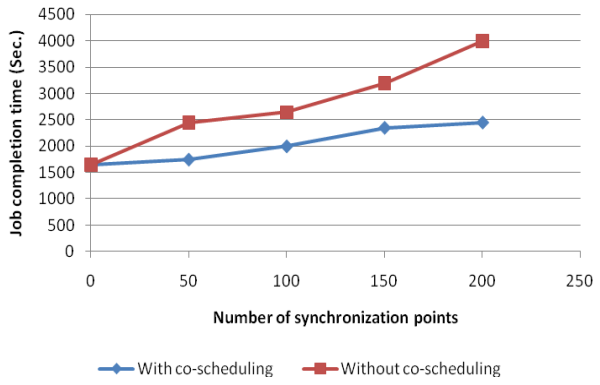

Fig. 3. Completion times of a job comprising of four tasks with two tasks co-working and running on a single virtual machine.

In the second experiment, we simulated the case in which the same job with the same task length and dependency was executed on two VMs. In this case, the two correlated tasks were located on two different VMs. Therefore, having known about this relation between the two dependant tasks and using co-scheduling to map the relevant virtual processors at the same time, unnecessary process blocks were avoided.

In the second experiment, each VM included two virtual processors. The hardware configuration was like the previous experiment, i.e. two processors, each capable of running one million instructions per second.

The results of simulating the second case are illustrated in Fig. 4. As in the first experiment, co-scheduling led to higher system performance as the number of synchronization points in correlated tasks increased.

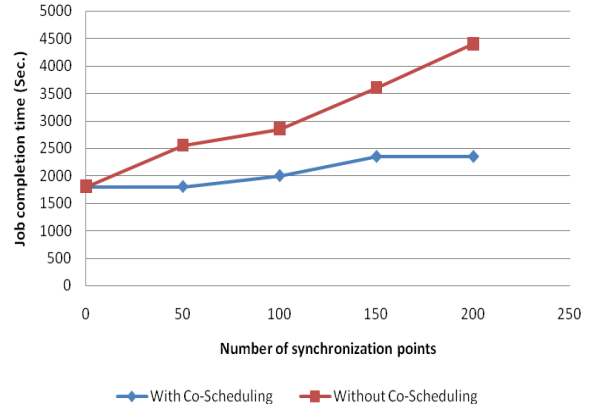

Fig. 4. Completion times of a job comprising of four tasks with two tasks co-working and running on two co-located virtual machines.

The results of simulation of both aforementioned cases in a concurrent virtual environment using co-scheduling technique showed a notable reduction in total execution time. It should be however noted that we have ignored the cost of finding co-scheduled processes in our simulations; this will indeed entail extra overhead in a real executing environment.

VI. Related Work

There are many techniques to schedule parallel applications on multi-processor architectures. As an example, backfilling [14] technique schedules parallel jobs that are at the end of ready queue to unutilized resources, instead of keeping resources idle. As another example, co-scheduling schedules concurrent and dependant processes or threads on different processors simultaneously.

The current modern hypervisors such as Xen [15] do not attempt to co-schedule virtual CPUs. The default scheduler on Xen, which is called credit scheduler, is a proportional share scheduler that tries to maximize system throughput while guaranteeing fairness.

There are also a handful of methods that try to optimize the efficiency of VMM scheduler. As an example, task-aware VM scheduling [16] infers the I/O bound nature of tasks and correlated incoming events with I/O-bound tasks. Other researches [17] have tried to exploit users' input in scheduling interactive VMs. Another relevant research [18] divides VM into two categories, batch and concurrent and then uses a proper scheduler for each one.

In contrast to the mentioned related researches that have sufficed to study a special case of VM scheduling, in this paper we have tried to highlight the main benefits and concerns of scheduling VMs. In addition, as an optimization, we proposed processor co-scheduling on VMs.

VII. Conclusion

In this paper, we studied VM scheduling in a Cloud environment from three different viewpoints: main challenges, scheduling advantages and a special optimization. In the first part, we discussed about the challenges arise when trying to schedule a set of VMs. In the next part, we mentioned some scheduling advantages of using VMs in Cloud distributed environments, like better reservation. We also proposed an optimization technique, namely, virtual processor co-scheduling method. This technique was described and also its effect on system performance was evaluated through simulation. It was shown that a higher system performance is attainable when concurrent and dependant jobs are mapped to executing processors at the same time; i.e., are co-scheduled. We are furthering our research by implementing our co-scheduling technique in Xen in such a way to find correlated tasks in a full virtualized case too.

References

[1] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, "Cloud computing," *Technical Report, IBM High Performance on Demand Solutions*, 2007.

[2] A. Batat and D. Feitelson, "Gang scheduling with memory considerations," In *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, Cancun, Mexico, 2000, pp. 109-114.

[3] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: challenges and opportunities," In *Proceedings of the 7th IEEE High Performance Computing and Simulation Conference, Germany*, 2009, pp. 1-11.

[4] R. N. Calheiros, R. Ranjan, C. De Rose, and R. Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," *Technical Report, GRIDS-TR-2009-1, Grid Computing and Distributed Systems Laboratory*, The University of Melbourne, Australia, 2009.

[5] P. Hellinckx , S. Verboven , F. Arickx, and J. Broeckhove, "Predicting parameter sweep jobs: from simulation to Grid implementation," In Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems, Japan, 2009, pp. 402-408.

[6] V. S. Adve and M. K. Vernon, "Parallel program performance prediction using deterministic task graph analysis," *ACM Transactions on Computer Systems*, vol. 22, no. 1, pp. 94-136, Feb. 2004.

[7] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," In *Proceedings of the 17th International Symposium on High Performance Distributed Computing, Boston*, MA, USA, 2008, pp. 87-96.

[8] M. Zhao and R. J. Figueiredo, "Experimental study of virtual machine migration in support of reservation of cluster resources," In *Proceedings of the 3rd International Workshop on Virtualization Technology in Distributed Computing*, Reno, Nevada, 2007.

[9] A. Moallem and S. A. Ludwig, "Using artificial life techniques for distributed Grid job scheduling," In *Proceedings of the ACM Symposium on Applied Computing*, Hawaii, 2009, pp. 1091-1097.

[10] D. Deb, M. M. Fuad, and M. J. Oudshoorn., "Toward autonomic distribution of existing object-oriented programs," In *Proceedings of the International Conference on Autonomic and Autonomous Systems*, Santa Clara, 2006.

[11] J. Luo and N. K. Jha, "Battery-aware static scheduling for distributed real-time embedded systems," In *Proceedings of the 38th Annual Design Automation Conference*, Las Vegas, Nevada, United States, 2001, pp. 444 - 449.

[12] VMware Workstation Product. http://www.vmware.com/products/ws/

[13] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. Lara, M. Brudno, and M. Satyanarayanan, "SnowFlock: rapid virtual machine cloning for Cloud computing," In *Proceedings of the 4th ACM European Conference on Computer systems*, Nuremberg, Germany, 2009, pp. 1-12.

[14] E. Shmueli and D. Feitelson, "Backfilling with lookahead to optimize the performance of parallel job scheduling," In *Proceeding of the 9the Workshop on Job Scheduling Strategies for Parallel Processing*, Washington, USA, 2003, pp. 228-251.

[15] 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 ACM Symposium on Operating Systems Principles*, New York, NY, USA, 2003, pp. 164-177.

[16] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-aware virtual machine scheduling for I/O performance," In *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environment, Washington*, DC, USA, 2009, pp. 101-110.

[17] B. Lin, P. A. Dinda, and D. Lu, "User-driven scheduling of interactive virtual machines," In *Proceedings of the 5th IEEE/ACM International Workshop on Grid, Washington*, DC, USA, 2004, pp. 380-387.