Virtualization & Scheduling in Real Time OS: A Survey

Shabnam Kumari
Asst. Prof, Department of CSE,
Sat Kabir Institute of Technology
& Management, Bahadurgarh,
Haryana, India

Reema
Asst. Prof, Department of CSE,
Sat Kabir Institute of Technology
& Management, Bahadurgarh,
Haryana, India

Shahdab Payami
M Tech Scholar, Department of
CSE, Sat Kabir Institute of
Technology & Management,
Bahadurgarh, Haryana, India

ABSTRACT

virtualization is the creation of a virtual (rather than actual) version of something, such as a hardware platform, operating system (OS), storage device, or network resources. In a virtualized environment, computing environments can be produced in a forceful dynamic manner, enlarged, become smaller or go in a specified direction or manner as demand varies. Virtualization is therefore highly suitable to a dynamic cloud infrastructure, because it provides important advantages in isolation, manageability and sharing.

KEYWORD: Virtualization, Virtual Machine Scheduler, Scheduling, Co-scheduling

INTRODUCTION

In computing, virtualization is the creation of a virtual (rather than actual) version of something, such as a hardware platform, operating system (OS), storage device, or network resources.

While a physical computer in the classical sense is clearly a complete and actual machine, both subjectively (from the user's point of view) and objectively (from the hardware system administrator's point of view), a virtual machine is subjectively a complete machine (or very close), but objectively merely a set of files and running programs on an actual, physical machine (which the user need not necessarily be aware of).

Virtualization and cloud computing are transforming IT. Cloud computing leverages virtualization to enable a more scalable and elastic model for delivering IT services. As a result, enterprises gain a more agile and efficient IT environment that is better able to respond to business needs. The principal driver behind the rapid adoption of virtualization has been cost reduction through server and other infrastructure consolidation. With virtualization, enterprises are no longer restricted to the traditional ratio of 1:1:1 for servers, operating systems and applications. Applications abstracted from the infrastructure enable IT to turn underutilized infrastructure into an elastic, resilient and secure pool of compute resources available to users on demand.

Global IT organizations have been quick to adopt virtualization to achieve cost benefits. By replacing physical IT assets with virtual resources, organizations are achieving up to 60 percent savings in capital expenses in their datacenters. According to IDC, server virtualization has become a standard feature in datacenter environments over the last few years, with virtual machine deployment outnumbering physical server shipments in 2009.

A virtualization platform with embedded management capabilities, such as high availability, automated load balancing and fault tolerance, reduces infrastructure complexity. Through automation and built-in manageability, it eliminates many time-consuming manual management tasks, significantly driving down operational IT costs. Cost reduction, however, is proving to be just one of the many benefits of virtualization.

It is the technique that removes linking together the hardware and operating system. It directs to the source of the logical resources abstraction away from their physical resources to be more flexible, reduce costs and make a good improvement in business value.

Essentially virtualizations in cloud have so many different types, such as, network virtualization, server virtualization and storage virtualization. Server virtualization can be described as an associating of single physical resources to several logical partitions or representations. In a virtualized environment, computing environments can be produced in a forceful dynamic manner, enlarged, become smaller or go in a specified direction or manner as demand varies. Virtualization is therefore highly suitable to a dynamic cloud infrastructure, because it provides important advantages in isolation, manageability and sharing.

II. TYPES OF VIRTUALIZATION

2.1. Hardware virtualization

Hardware virtualization or platform virtualization refers to the creation of a virtual machine that acts like a real computer with an operating system. Software executed on these virtual machines is separated from the underlying hardware resources. For example, a computer that is running Microsoft Windows may host a virtual machine that looks like a computer with the Ubuntu Linux operating system; Ubuntu-based software can be run on the virtual machine.

In hardware virtualization, the host machine is the actual machine on which the virtualization takes place, and the guest machine is the virtual machine. The words host and guest are used to distinguish the software that runs on the physical machine from the software that runs on the virtual machine. The software or firmware that creates a virtual machine on the host hardware is called a hypervisor or Virtual Machine Manager.

Different types of hardware virtualization include:

1. Full virtualization: Almost complete simulation of the actual hardware to allow software, which typically consists of a guest operating system, to run unmodified.

2. Partial virtualization: Some but not all the target environment is simulated. Some guest programs, therefore, may need modifications to run in this virtual environment.

3. Para virtualization: A hardware environment is not simulated; however, the guest programs are executed in their own isolated domains, as if they are running on a separate system. Guest programs need to be specifically modified to run in this environment.

Hardware-assisted virtualization is a way of improving the efficiency of hardware virtualization. It involves employing specially designed CPUs and hardware components that help improve the performance of a guest environment.

Hardware virtualization is not the same as hardware emulation. In hardware emulation, a piece of hardware imitates another, while in hardware virtualization; a hypervisor (a piece of software) imitates a particular piece of computer hardware or the entire computer. Furthermore, a hypervisor is not the same as an emulator; both are computer programs that imitate hardware, but their domain of use in language differs.

2.2. Desktop Virtualization

Desktop virtualization is the concept of separating the logical desktop from the physical machine. One form of desktop virtualization, virtual desktop infrastructure (VDI), can be thought as a more advanced form of hardware virtualization. Rather than interacting with a host computer directly via a keyboard, mouse, and monitor, the user interacts with the host computer using another desktop computer or a mobile device by means of a network connection, such as a LAN, Wireless LAN or even the Internet. In addition, the host computer in this scenario becomes a server computer capable of hosting multiple virtual machines at the same time for multiple users.

- Software virtualization
- Storage virtualization
- Memory virtualization

III. VIRTUAL MACHINES

A virtual machine (VM) is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine. Virtual machines are separated into two major categories, based on their use and degree of correspondence to any real machine:

1. A system virtual machine provides a complete system platform which supports the execution of a complete operating system (OS). These usually emulate an existing architecture, and are built with either the purpose of providing a platform to run programs where the real hardware is not available for use (for example, executing software on otherwise obsolete platforms), or of having multiple instances of virtual machines lead to more efficient use of computing resources, both in terms of energy consumption and cost effectiveness (known as hardware virtualization, the key to a cloud computing environment), or both.

2. A process virtual machine (also, language virtual machine) is designed to run a single program, which means that it supports a single process. Such virtual machines are closely suited to one or more programming languages and built with the purpose of

providing program portability and flexibility (amongst other things). An essential characteristic of a virtual machine is that the software running inside is limited to the resources and abstractions provided by the virtual machine - it cannot break out of its virtual environment.

IV. PROBLEM DEFINITION AND DIRECTION FOR PROPOSED SOLUTION

A discussion towards the RESOURCE MANAGEMENT of VIRTUAL MACHINES in CLOUD and how to make resources more efficiently available to clients, is provided. The notion of JOB SCHEDULING is addressed.

4.1. Deadline Aware Virtual Machine Scheduler

A novel approach to optimize job deadlines when run in virtual machines by developing a deadline-aware algorithm that responds to job execution delays in real time, and dynamically optimizes jobs to meet their deadline obligations.

Performance Metrics

To access the performance of our scheduling algorithm, the following metrics are defined in the system:

- System performance to measure total number of jobs completed during a period of time.
- Deadline miss rate representing the number of jobs missing their deadline, thus being terminated by the scheduler.
- Utilization rate for the CPU and memory to measure how long each resource have been active.

Questions to be answered

- How dynamic scheduling of workloads at the machine level will work once the batch system have scheduled a job to a particular node given that job will be executed in a virtual machine container?
- What kind of scheduling technique could be used to optimize multiple virtual machines running HPC jobs?
- Which parameter of the job could be considered as pivotal in scheduling policy; deadline-duration ratio or failure rate? Or both of them could be part of the same scheduling technique?
- What would be the mix of executing job types on the machine to increase resource utilization?

4.2. Utility Based Job Scheduler

Cloud computing system is featured by its workload, deadline and corresponding utility obtained by its completion before deadline, which also are factors considered in deriving an effective scheduling algorithm. Utility is attained from completed jobs. So amount of utility that cloud system obtained is defined in terms of completion of jobs.

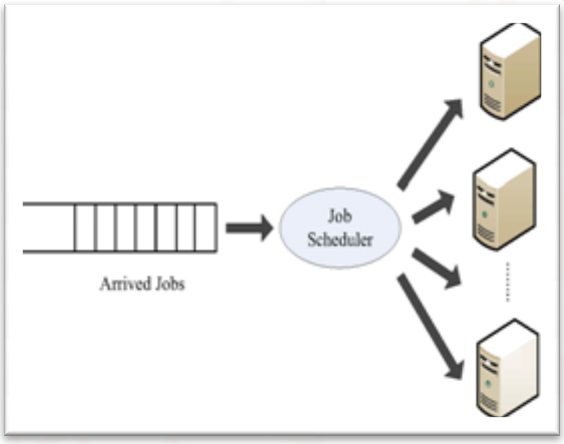

Figure 1.1: Job Scheduler [7]

4.3. Dynamic Virtual Machine Job Scheduler

To serve to diverse user communities with often competing Quality of Service (QoS) requirements for their jobs/virtual machines, some jobs being more CPU or memory intensive than the others and vice versa, requires a dynamic and intelligent resource scheduling which is adaptive as the nature of workloads at any given moment changes. QoSvaries from different utility context such as its different for EC2 user community.

X factor is a ratio of job I that is projected to miss its current deadline and is determined by

Xi = (job duration remaining-time to deadline) / job duration remaining. It uses three approaches for determining the X value.

Using a static threshold value

- Using a threshold value which adapts according to the existing conditions
- Using cdf

4.4. New Metrics of Job Scheduling Of Virtual Machines

In this paper, there are a new set of metrics, called potential capacity (PC) and equilibrium capacity

(EC), of resources that incorporate these dynamic, elastic, and sharing aspects of co-located virtual machines. Then show the mesh of this set of metrics smoothly into traditional scheduling algorithms.

V. SCHEDULING

Scheduling mechanism is the most important component of a computer system. Scheduling is the strategy by which the system decides which task should be executed at any given time. There is difference between real-time scheduling and multiprogramming timesharing scheduling. It is because of the role of timing constraints in the evaluation of the system performance.

5.1. Types of Scheduling

Three types of scheduler: Long-term, medium-term and short-term.

5.1.1. Long-term Scheduling:

When memory is available, which task (job) needs to be selected to be fetched and serviced as a process? The issue here is memory residency. As long as the process is active, it would be residing in the primary memory.

Long-term schedule is about

- New
- Exited

Long-term scheduling dictates degree of multiprogramming. If too high, system performance drops. If too low, throughput would be low.

5.1.2. Medium-term Scheduling:

Which and when processes are to be suspended and resumed?

Medium-term scheduling is concerned about

- Blocked – Suspended
- Ready – Suspended

It is concerned with memory management. It is vital that it interacts with the dispatcher unit efficiently to provide good system performance.

5.1.3. Short-term Scheduling (dispatching):

Which of the ready processes (in the CPU queue) is to have CPU resources and for how long?

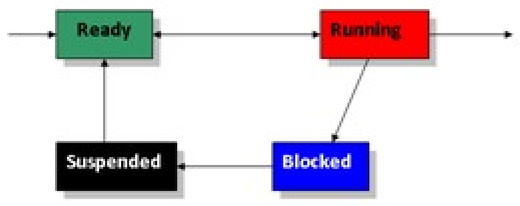

Figure 1.2: States of a Process

Dispatching is concerned about

- Running
- Ready
- Blocked

Short-term scheduling or dispatching is concerned with the allocation of CPU to processes to meet some pre-defined system performance objectives.

5.2. Cloud Scheduling

The main reason of using a Cloud system is to improve the performance of large-scale application programs, high scalability and fault tolerance. Therefore it is essential to keep the cost of creating and managing parallelism as low as possible.

As for scheduling in a Cloud system, there are three goals to lower down the cost:

- Good processor utilization:

All processors have work in their queues at all times. All processors, which have tasks assigned to them from the same program finish execution at the same time, thus the user gets the expected speedup. Processors spend most of their time doing useful work rather than coordination the division of work.

- Good synchronization effectiveness:

Tasks are scheduled in such a way that interacting tasks across with fine-grained interaction should be running at the same time.

- Low communication/memory-access cost:

Tasks are scheduled in such a way that communication time, either message passing or shared-memory latency is accounted for, and minimized. Scheduling data structures should be arranged so that they are no a source of contention.

5.3. Single Shared Ready Queue

It is a simple approach. A single global queue shared by all the processors in the system. Whenever a processor is available and the queue is not empty, the processor will be scheduled with a new task. The scheduling policies such as First Come First Serve (FCFS) and Shortest Job First (SJF) can be easily implemented. [9] It seems as if this approach may yield good processor utilization. However, this approach does not provide any mean to schedule fine-grained interacting tasks to run at the same time. Consider the following scenario: Task A already scheduled for a processor, but it mush wait for completion of Task B, which is still in the queue. Even though Task A keeps the processor busy, all it does is to wait. This approach also has potential high communication/memory access cost, because this global queue occupies a region of memory that can be accessed by all the processors simultaneously.

5.4. Co-scheduling

In co-scheduling, all runable tasks of an application are scheduled on different processors simultaneously. Without coordinated scheduling, the processes constituting parallel tasks may suffer communication latencies because of processor thrashing. In recent years, researchers have developed parallel scheduling algorithms that can be loosely organized into three main classes, according to the degree of coordination between processors: explicit or static co-scheduling, local scheduling, and implicit or dynamic co-scheduling.

5.4.1. Explicit Co-scheduling

Explicit co-scheduling ensures that the scheduling of communication jobs is coordinated by creating a static global list of the order in which jobs should be scheduled and then requiring a simultaneous context-switch across all processors. This may be accomplished statically by agreeing upon a global schedule in advance or dynamically by having a "master" local scheduler direct other schedulers by communicating with them at each context switch. However, this approach has quite a few drawbacks. Since it requires identifying parallel tasks in the application in advance, it complicates the implementation. Furthermore, it interacts poorly with interactive use and load imbalance.

5.4.2. Local Co-scheduling

Conversely, local scheduling allows each processor to independently schedule its processes. Although attractive due to its ease of construction, the performance of fine-grain communicating jobs degrades significantly because scheduling is not coordinated across processor

5.4.3. Implicit Co-scheduling

An intermediate approach, implicit co-scheduling, allows each of the local schedulers to make decisions independently, but relies on local schedulers to take the communication behavior of local processes into account when making decisions. Local schedulers can converge on co-scheduling behavior since each sees similar or related communication behavior by local processes that are part of parallel applications. There are many forms of implicit co-scheduling developed over the years.

5.4.4. D_EDF

Deadline Monotonic is fixed-priority scheduling algorithm whereas EDF is dynamic-priority algorithm. A fixed-priority scheduler assigns the same priority to all instances of the same task, thus the priority of each task is fixed with respect to other tasks. However, a dynamic-priority scheduler may assign different priorities to different instances of the same task, thus the priority of each task may change with respect to other tasks as new task instances arrive and complete.

REFERENCES

[1] P. A. Laplante, Real-Time Systems Design and Analysis, 3rd ed., IEEE Press, 2004.

[2] G. C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, 3rd ed., Springer, 2005.

[3] R. L. Panigrahi and M .K. Senapaty, "Real Time System for Software Engineering: An Overview", Global Journal for Research Analysis, Vol. 3, Issue 1, pp. 25-27, January 2014.

[4] J. W. S. Liu, Real-Time Systems, Prentice Hall, 2000.

[5] A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, 8th ed., Wiley, 2012.

[6] R. J. Creasy, "The Origin of the VM/370 Time-Sharing System," IBMJ. Res. Dev., vol. 25, no. 5, pp. 483–490, 1981.

[7] G. J. Popek and R. P. Goldberg, "Formal Requirements for Virtualizable

Third Generation Architectures," Commun.ACM, vol. 17, no. 7, pp.412–421, 1974.

[8] Y. Wang, J. Zhang, L. Shang, X. Long, and H. Jin, "Research of Realtime Task in Xen Virtualization Environment," in ICCAE'10, 2010.

[9] T. Gaska, B. Werner, and D. Flagg, "Applying Virtualization to Avionics Systems - The Integration Challenges," in DASC'10, 2010.

[10] D. Patnaik, A. Bijlani, and V. Singh, "Towards High-Availability for IP Telephony Using Virtual Machines," in IMSAA'10, 2010.

[11] D. Patnaik, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Performance Implications of Hosting Enterprise Telephony Applications on Virtualized Multi-Core Platforms," in IPTComm'09, 2009.

[12] P. Goyal, X. Guo, and H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," in OSDI'96, 1996.

[13] Z. Deng and J. W.-S.Liu, "Scheduling Real-time Applications in an Open Environment," in RTSS'97, 1997.

[14] J. Regehr and J. A. Stankovic, "HLS: A Framework for Composing Soft

Real-Time Schedulers," in RTSS'01, 2001.

[15] S. Kato, R. Rajkumar, and Y. Ishikawa, "A Loadable Real-Time Scheduler S ite for Multicore Platforms," Technical Report CMUECE- TR09-12, 2009.[Online]. Available: http://www.contrib.andrew.cmu.edu/»shinpei/papers/techrep09.pdf