RESEARCH ARTICLE OPEN ACCESS

Two Level Scheduler for Cloud Computing Environment

Sheeja Y.S.*, Dancy Kurian**, Kala Karun***
**(Department of Computer Science, College of Engineering Attingal, Kerala, India.*
Email: yssheeja@gmail.com)
*** (Department of Computer Science, College of Engineering Attingal, Kerala, India.*
Email: dancyk@gmail.com)
**** (Department of Computer Science, College of Engineering Kottarakara, Kerala, India.*
Email: kalavipin@gmail.com)

ABSTRACT
Cloud computing delivers infrastructure, platform, software and other applications as services, which are made available as subscription-based services in a pay-as-you-go model to consumers. Quantifying the performance of scheduling and allocation policy on Cloud infrastructures like hardware, software, services for different application and service models under varying load, energy performance such as power consumption, heat dissipation, and system size is an extremely challenging problem. This paper presents the implementation of an efficient Quality of Service based Meta-Scheduler and Backfill strategy based light weight Virtual Machine Scheduler for dispatching jobs. The user centric Meta-scheduler deals with selection of proper resources to execute high level jobs. The system centric Virtual Machine scheduler optimally dispatches the jobs to processors for better resource utilization.

Keywords **-** Back filling, Cloud Computing, Meta-scheduler, Quality of service, Virtual Machine.

Date of Submission: 27-11-2017 Date of acceptance: 08-12-2017

I. INTRODUCTION

Cloud computing is a cost effective model for providing services and it makes IT management easier and more responsive to the changing needs of the business[1]. Cloud computing can be defined as "a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers"[1]. Cloud computing is a type of parallel and distributed system. Job scheduling problem is a core and challenging issue in cloud computing. It is impossible to predict the job execution time in cloud environment. As Cloud computing is a rapidly evolving research area, there is a severe lack of defined standards, tools and methods that can efficiently tackle the infrastructure and application level complexities. Hence in the near future there would be a number of research efforts both in academia and industry towards defining core algorithms, policies; application benchmarking based on execution contexts. The access to the infrastructure incurs payments in real currency in cloud environment. The simulation based approaches provide significant benefits, as it allows researchers to test their proposed algorithms and protocols in a repeatable and controlled environment free of cost, and to find solution to the performance bottlenecks before deploying in the real cloud [2]. By extending the basic functionalities already exposed by CloudSim, researchers would be able to perform tests based on specific scenarios and configurations, hence allowing the development of best practices in all the critical aspects related to Cloud Computing. The CloudSim toolkit supports First Come First Serve (FCFS) and Round Robin (RR) scheduling strategies for internal scheduling of jobs. FCFS and RR suffer from long average waiting time for longer jobs which necessitates for the deployment of a better scheduling strategy at the cluster level. So here use a scheduling algorithm based on backfilling which allows smaller jobs to move forward in the schedule as long as such movement does not cause any other scheduled jobs to be further delayed. This will reduce the waiting time of longer jobs. Clouds aim to power the next generation datacenters by architecting them as a network of virtual services (hardware, database, user-interface, application logic) so that users are able to access and deploy applications from anywhere in the world on demand at competitive costs depending on users QoS (Quality of Service) requirements .

II. PROBLEM DEFINITION

There are various scheduling techniques available for the scheduling of jobs in cloud computing. The CloudSim [3] toolkit supports First Come First Serve (FCFS) and Round Robin (RR) scheduling strategies for internal scheduling of jobs. FCFS and RR suffer from long average waiting time for longer jobs which necessitates for the deployment of a better scheduling strategy at the cluster level. Back filling scheduling policies allows smaller jobs to move forward in the schedule as long as such movement doesn't cause any other scheduled job to be further delayed.

This work concentrates on the design of a system that schedules various types of jobs in cloud environment. The activities involved in job scheduling for cloud environment includes the selection of processing resource like datacenter, host and virtual machine and the processing order of jobs(cloudlets) for every resource. Some of the constraints to be considered for scheduling include the QoS specifications like deadline, budget, and software licenses of jobs, job dependencies and resource limitations. The proposed two- level scheduler focuses on optimizing the system throughput by maximizing the overall resource utilization and guaranteeing increased performance of the applications. The proposed approach extends the CloudSim toolkit [2], by implementing a novel high-level meta-scheduler. The Meta scheduler selects proper datacenter based on customer requirements like deadline and budget. As meta-scheduler cannot have a control over the resources at a datacenter and the full set of jobs submitted to the resources, implemented a low-level local scheduler to perform efficient job scheduling in cloud environment. This low level scheduler is designed based on backfilling concept. The simple VM Provisioner of the CloudSim chooses the host with less PEs in use, as the host for VM. This heuristics ensures load balancing. Nevertheless, many VM Create Requests fail, even though the required numbers of free PEs are available across various hosts. This paper also modified the simple VM provisioner to optimal VM provisioner.

III. LITERATURE SURVEY

The default algorithms used by current batch job schedulers for parallel supercomputers are all rather similar to each other. In essence, they select jobs for execution in first- come-first-serve (FCFS) order, and run each job to completion. The problem is that this simplistic approach causes significant fragmentation, as jobs do not pack perfectly and processors are left idle [4]. Most schedulers therefore use backfilling: if the next queued job cannot run because sufficient processors are not available, the scheduler nevertheless continues to scan the queue, and selects smaller jobs that may utilize the available resources.

3.1 Scheduling Algorithms

General concept of backfilling allows smaller jobs to move forward in the schedule as long as such movement does not cause any other scheduled jobs to be further delayed. This section discusses some of the variants of the Backfilling scheduling strategies that can be used at the cluster level. In EASY (Extensible Argonne scheduling sYstem) backfilling, only the first queued job is given Earliest Start Time.[6] Now it is possible to schedule and dispatch the smaller jobs if they would not delay the start of the job in the head of the waiting queue. In the second approach namely, Conservative Backfilling every queued job is given guaranteed start time, so that it has a bounded delay [5]. The third approach namely, Slack based backfilling differs from conservative method by supporting priorities. It assigns each waiting job some slack, which measures the maximal amount of time that the job may be delayed beyond its initially assigned start time. When a job is delayed or speeds up its slack changes accordingly. This way the scheduler enjoys more flexibility than conservative scheduling, but still retains the execution guarantee. The conservative backfilling achieves the same result as the slack based method, but it is comparatively light weight. Hence this proposed work implements conservative backfilling at the cluster level for better throughput.

3.1.1 EASY (Extensible Argonne scheduling System) Backfilling

Backfilling requires the runtime of jobs to be known: both when computing the reservation (requires knowing when processors of currently running jobs will become available) and when determining if waiting jobs are eligible for backfilling (must terminate before the reservation). Therefore, EASY required users to provide a runtime estimate for all submitted jobs and the practice continues to this day. Jobs that exceed their estimates are killed, so as not to violate subsequent commitments. The assumption is that users would be motivated to pro-vide accurate estimates, because jobs would have a better chance to backfill if their estimates are tight, but would be killed if they are too short [6].

In EASY backfilling, the scheduler may backfill later jobs even if that delays the expected start time of other jobs, so long as the first jobs expected start time isn't delayed. EASY backfilling selects a small job to backfill if it does not delay the start time of the first job in the queue. The resource utilization is improved. The requirement of user-estimated run-time of jobs is lower. The small jobs

will be able to get more opportunities for backfilling. It is more flexible to backfill. However, the large job may be delayed to run more easily. The main drawback is the wide jobs get reservation only if they are at front of the queue.

3.1.2 Conservative Backfilling

A key benefit of Conservative Backfilling is that each job is granted a guaranteed starting time when it is submitted. (It may start earlier, but will not be delayed later than this time.) These guarantees lead Conservative Backfilling to benefit wide jobs, jobs requiring many processors, relative to other backfilling strategies From a fairness standpoint, this guarantee ensures that wide or long jobs, which are less likely to benefit from backfilling, are not harmed by jobs that backfill more easily. These guarantees also make the scheduler more predictable since each user has a bound on when their jobs will run [5] [6].

Conservative backfilling maintains a profile containing a tentative schedule for all jobs. When a job arrives, it is placed in the earliest possible spot within the profile, i.e. it is scheduled to start at the earliest time that does not disturb any previously placed job. The only other profile changes occur when a job finishes early, creating a "hole" that potentially allows other jobs to move earlier. In this case, Conservative initiates compression, the re-examination of each job in the order of its current starting time in the profile. Each job is removed from the schedule and then reinserted at the earliest possible time. Compression never delays a job since the job can always fit back into the profile at the same spot, but some jobs move earlier, into a hole or spaces vacated by jobs that have moved. Since no job's planned start time is ever delayed, each job's initial reservation is an upper bound on its actual starting time. EASY and Conservative Backfilling use First-Come-First-Serve (FCFS) order. The disadvantage is that it is unnecessary to provide reservation to all jobs whether it is truly needed or not. This will reduce the backfilling effect.

3.1.3 Slack Based Backfilling

This approach is based on conservative backfilling but relaxes it by permitting constrained delays, called slack. The goal is to increase utilization in high-load phases and subsequently response times by better packing, while keeping the schedule close to FCFS. The low level local scheduler is implemented using conservative backfilling algorithm. When a job is delayed or speeds up its slack changes accordingly. This way the scheduler enjoys more flexibility than conservative scheduling, but still retains the execution guarantee [6]. The conservative backfilling achieves the same result as the slack based method, but it is comparatively light weight. Hence our proposed work implements conservative backfilling at the cluster level for better throughput.

3.2 Cloud Simulator-CloudSim

The access to real cloud infrastructure incurs payments in real currency in cloud environment. The simulation based approaches provide significant benefits, as it allows researchers to test their proposed algorithms and protocols in a repeatable and controlled environment free of cost, and to find solution to the performance bottlenecks before deploying in the real cloud [2].

3.2.1 Modeling CloudSim

The core hardware infrastructure services related to the Clouds are modelled in the simulator by a Datacenter component for handling service requests. These requests are application elements sandboxed within VMs, which need to be allocated a share of processing power on Datacenter's host components. By VM processing, it means a set of operations related to VM life cycle: provisioning of a host to a VM, VM creation, VM destruction, and VM migration.

A Datacenter is composed by a set of hosts, which is responsible for managing VMs during their life cycles. Host is a component that represents a physical computing node in a Cloud: it is assigned a pre-configured processing (expressed in millions of instructions per second –MIPS, per CPU core), memory, storage, and a scheduling policy for allocating processing cores to virtual machines. The Host component implements interfaces that support modeling and simulation of both single-core and multi-core nodes.

Allocation of application-specific VMs to Hosts in a Cloud-based data center is the responsibility of the Virtual Machine Provisioner component. This component exposes a number of custom methods for researchers, which aids in implementation of new VM provisioning policies based on optimization goals (user centric, system centric). The default policy implemented by the VM Provisioner is a straightforward policy that allocates a VM to the Host in First-Come-First-Serve (FCFS) basis. The system parameters such as the required number of processing cores, memory and storage as requested by the Cloud user form the basis for such mappings. Other complicated policies can be written by the researchers based on the infrastructure and application demands.

For each Host component, the allocation of processing cores to VMs is done based on a host allocation. The policy takes into account how many processing cores will be delegated to each VM, and how much of the processing core's capacity will effectively be attributed for a given VM. So, it is

possible to assign specific CPU cores to specific VMs (a space-shared policy) or to dynamically distribute the capacity of a core among VMs (time-shared policy), and to assign cores to VMs on demand, or to specify other policies.

3.2.2. Modeling VM Allocation

One of the key aspects that make a Cloud computing infrastructure different from a Grid computing is the massive deployment of virtualization technologies and tools. Hence, as compared to Grids, we have in Clouds an extra layer (the virtualization) that acts as an execution and hosting environment for Cloud-based application services.

Hence, traditional application mapping models that assign individual application elements to computing nodes do not accurately represent the computational abstraction which is commonly associated with the Clouds. For example, consider a physical datacenter host that has single processing core, and there is a requirement of concurrently instantiating two VMs on that core. Even though in practice there is isolation between behaviors (a context) of both VMs, the amount of resources available to each VM is constrained by the total processing power of the host. This critical factor must be considered during the allocation process, to avoid creation of a VM that demands more processing power than the one available in the host, and must be considered during application execution, as task units in each virtual machine shares time slices of the same processing core.

To allow simulation of different policies under different levels of performance isolation, CloudSim supports VM scheduling at two levels: First, at the host level and second, at the VM level. At the first level, it is possible to specify how much of the overall processing power of each core in a host will be assigned to each VM. At the next level, the VMs assign specific amount of the available processing power to the individual task units that are hosted within its execution engine.

IV. DESIGN

The Development is divided into three modules. Fig.1 shows the components in cloud. Cloud computing environment can be virtualized as a collection of n datacenters. Upon which n hosts can be created and each host may contain m virtual machine with processing elements.

4.1 Meta Scheduler

In CloudSim, Datacenter Broker component randomly selects the datacenter irrespective of their heterogeneity in hardware, software configuration and pricing schemes for usage. Then the broker maps the submitted cloudlets to the created virtual machines in a circular fashion without considering the Processing Elements (PEs) required by the cloudlets.

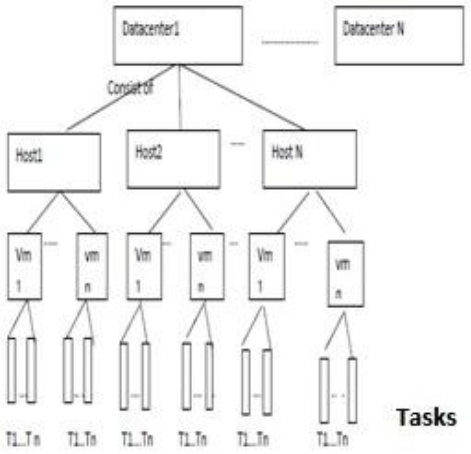

Fig. 1 Cloud Computing Components.

The proposed meta-scheduler that selects the datacenter based on user defined QoS specifications such as deadline and budget. For example if the user requirement is budget constrained, it tries to create as many VMs as possible in a datacenter which has reduced price and the remaining VMs in the other datacenters. Further the meta-scheduler identifies a VM with sufficient number of

Virtual CPUs (VCPUs) before mapping the cloudlet to VM and thus failure in cloudlet execution is avoided.

Algorithm

Step1.Create Datacenters

Step2.Create processing elements (PEs) and host in the datacenters created in step1

Step3. Create a Datacenter Characteristics object that stores the properties of a data center: architecture, OS, list of Machines, allocation policy: time- or space-shared, time zone and its price (G$/Pe time unit).

Step4. Submit the cloudlet (cloud application) that consists of the requirements of customer specification like the budget and deadline.

Step5.Compare the parameters in step 4 with those were in datacenter and select an appropriate datacenter.

4.2 Low Level Local Scheduler

The existing space shared local scheduler in CloudSim employs simple FCFS Policy. It is associated with each VM, which queues the newly arrived cloudlets, in case of non availability of required resources. When resources become free,

only newly arrived cloudlets are served, but not the queued ones, hence they suffer from starvation.

The proposed Intra VM Scheduler uses Modified Conservative Backfilling method as the queuing policy. In this every queued job is given a guaranteed Earliest Start Time (EST) and the newly arrived cloudlets are backfilled, only when it does not affect the EST of the already queued cloudlets. This hard guarantee eliminates starvation of queued cloudlets also respecting FCFS. This is one of the typical requirements for real time jobs.

Algorithm

1. Create a datacenter with one host
2. Create appropriate number of PEs in the host.
3. Submit cloudlets to Datacenter Broker with different arrival time
4. For each cloudlet in the queue

If the number of free PEs are greater than required PEs and if the EST doesn't affect cloudlet at front. Schedule this for execution Else put to waiting queue

5. Repeat step4 until all the cloudlets have been scheduled

4.3 Optimal VM Provisioner

The simple VM Provisioner of the CloudSim chooses the host with less PEs in use, as the host for VM. This heuristics ensures load balancing. Nevertheless, many VM create requests fail, even though the required numbers of free PEs are available across various hosts.

The optimal VM Provisioner in the proposed system rectifies the said problem by optimally creating VMs in the hosts by ordering the request appropriately. The VM creation requests with more resources are allocated followed by the requests with fewer resources, thus minimizing the number of failures in VM creation.

Algorithm

1. Use any sorting method to sort the VM create request based on required number of PEs in ascending order.
2. Create VM in Host having required number of PEs.

V. RESULTS AND DISCUSSION

The Proposed system is tested for a number of cloudlets. The Meta scheduler selects appropriate data center with DCID 2 and the low level local scheduler selects virtual machine (VMID 0) with suitable number of processing elements. Then it schedules the cloudlets to virtual machine. The low level local scheduler is tested with eight cloudlets numbered from Cl 0 to Cl 7. Each cloudlet requires processing elements 2, 3, 2, 1, 2, 5,2and 2 respectively. The datacenter selected is Datacenter2 with VM0.The Table.1 shows the details of scheduling of cloudlets and execution. The columns in the table are cloud (Cl) ID, number of processing elements(PE), datacenter (DC) ID, virtual machine(VM) ID, execution(Ex) time, start time and finish time.

Table.1 Scheduling Details

Cl ID	No PE	DC ID	VM ID	Ex. Time	Start Time	Finish Time
0	2	2	0	800	0.1	800.1
3	1	2	0	400	800.1	1200.1
2	2	2	0	1200	80.1	1280.1
6	2	2	0	1200	1280.1	2480.1
4	2	2	0	2000	1200.1	3200.1
7	2	2	0	800	2480.1	3280.1
1	3	2	0	400	3280.1	4080.1
5	4	2	0	400	4080.1	4480.1

VI. CONCLUSION

The paper proposed the enhancement of the existing scheduling strategy in the cloud environment by proposing a two-level scheduler optimizing scheduler. The implementation of an efficient Quality of Service (QoS) based Meta-Scheduler and Backfill strategy based light weight Virtual Machine Scheduler for dispatching jobs presented in the paper. The User centric meta-scheduler deals with selection of proper resources to execute high level jobs. The system centric Virtual Machine (VM) scheduler optimally dispatches the jobs to processors for better resource utilization**.** In addition, the novel optimized VM Provisioner for enhanced resource utilization is also implemented. This scheduler can be extend to include inter VM scheduling.

REFERENCES

[1] Srikumar Venugopal , Rajkumar Buyya, An SCP-based Heuristic approach for Scheduling Distributed Data-Intensive Applications on Global Grids.*Journal of Parallel and Distributed Computing, Vol 68(4),* 2008.pp. 471-487

[2] Anton Beloglazo, Rajkumar Buyya,Energy Efficient Allocation of Virtual Machines in Cloud Data Centers, *Proceedings of the 10th IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGrid), Melbourne, Australia, 2010.

[3] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya, A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems, *Advances in Computers, Vol 82*, 2011.pp.47-111

[4] William Voorsluys, James Broberg, Rajkumar Buyya, *Introduction to Cloud Computing,Cloud Computing: Principles and Paradigms*, (Wiley Press, New York, USA).2011.pp.1-41.
[5] Mohsen Amini Salehi, Bahman Javadi, Rajkumar Buyya. QoS and Preemption aware Scheduling in Federated and Virtualized Grid Computing Environments, *Journal of Parallel and Distributed Computing (JPDC), Vol 72 (*2), 2012 Pages: 231-245.
[6] D. Lifka, "The ANL/IBM SP scheduling system". *In Proc of 1st Workshop on Job Scheduling Strategies for Parallel Processing* (JSSPP),*Vol 949*. 1995.pp. 295–303.

International Journal of Engineering Research and Applications (IJERA) is **UGC approved** Journal with Sl. No. 4525, Journal no. 47088. Indexed in Cross Ref, Index Copernicus (ICV 80.82), NASA, Ads, Researcher Id Thomson Reuters, DOAJ.

Sheeja Y.S "Two Level Scheduler for Cloud Computing Environment." International Journal of Engineering Research and Applications (IJERA) , vol. 7, no. 12, 2017, pp. 12-17.