A SURVEY ON ECONOMIC CLOUD SCHEDULERS FOR OPTIMIZED TASK SCHEDULING

S. Nagadevi¹, K. Satyapriya², Dr. D. Malathy³

Address for Correspondence
Department of Computer Science and Engineering, SRM University, Chennai, India

ABSTRACT
Cloud computing is computing paradigm where applications, data, memory, bandwidth and IT services are provided over the Internet. Cloud computing is purely based on pay per usage model. Cloud Service Providers (CSP) earns money by providing virtual resources to the Cloud Users (CU). The time during which resources are utilized is optimized by cloud scheduling algorithms. The Task management is the key role in cloud computing systems. Task scheduling problems are the premier which relate to the efficiency of the whole cloud computing facilities. Here, we have analyzed and compared the performance of various task scheduling algorithms for cloud environment.

KEYWORDS: Cloud, Task Scheduling, Optimization, CSP, CU, Virtual resources

1. INTRODUCTION
Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction[1]. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models. The five essential characteristics are on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. The three service models are Cloud Software as a Service (SaaS), Cloud Platform as a Service (PaaS), and Cloud Infrastructure as a Service (IaaS). The four deployment models are Private Cloud, Community Cloud, Public Cloud, and Hybrid Cloud. Underlying these services are data centers that provide "virtual machines" (VMs). Virtual machines make it easy to host computation and applications for large numbers of distributed users by giving each the illusion of a dedicated computer system. The cloud computing overview is shown Fig 1.

In this paper we have analyzed and compared the various task scheduling algorithms in order to evaluate and optimize the profit, resource usage and time. That different tasks cause overhead costs of resources which is not suitable for cloud [2]. This problem leads to over-costed and over-priced in some high volume simple tasks while under-costed and under-priced in low volume complex tasks.

B. Need for task Scheduling in cloud
The increasingly popular cloud model provides as-needed access to computing with the appearance of unlimited resources. Users are given access to a variety of data and software utilities to manage their work. Billing for services is based on usage; users rent virtual resources and pay for only what they use. The need for the resources (software and hardware) has been increased tremendously. Cloud service providers do business by servicing the users by granting their resources (memory, disk, bandwidth etc.). Providers and cloud users rent virtual resources and pay for only what they use. The competition of both sides of providers and users makes the scheduling problems more vital. A good scheduling algorithm improves the CPU utilization, turnaround time and cumulative throughput. A poor scheduling algorithm may result in terrible consequences, for example service providers lose money and even go out of business.

C. Economic Schedulers
Energy efficient economic schedulers are more suitable for cloud-based scheduling than traditional multiprocessor models. In economics, market-based and auction-based schedulers handle two main interests. Market-based schedulers are applied when a large number of naive users can not directly control service price in commodity trade. Mainstream cloud providers apply market-based pricing schemes in reality. The concrete schemes vary from provider to provider. As the most successful IaaS provider, Amazon EC2 supports commodity and posted pricing models for the convenience of users. Another alternative is auction-based scheduler, which is adapted to situations where a small number of strategic users seeking to attain a specific service compete with each other.

In auctions, users are able to commit the auction price. Amazon spot instance is an example of auction-based model. Instance price adjusts from time to time, depending on the supply and demand. As a result, users should estimate the future price and make its proposal in an auction before placing a spot instance request [3].

A summary of economic schedulers is shown in Table 1.
TABLE 1 ECONOMIC SCHEDULERS

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Economic model</th>
<th>Computing paradigm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster-on-demand</td>
<td>tendering</td>
<td>Cluster</td>
</tr>
<tr>
<td>Masix</td>
<td>commodity</td>
<td>Cluster</td>
</tr>
<tr>
<td>Stanford Peers</td>
<td>auction/bartering</td>
<td>Peer to peer</td>
</tr>
<tr>
<td>D'Agents</td>
<td>proportion shared auction</td>
<td>Mobile-agent</td>
</tr>
<tr>
<td>Faucets</td>
<td>tendering</td>
<td>Grid</td>
</tr>
<tr>
<td>Nimrod-G</td>
<td>commodity/auctions</td>
<td>Grid</td>
</tr>
<tr>
<td>Marketnet</td>
<td>posted price</td>
<td>Distributed information</td>
</tr>
<tr>
<td>Cloudbus</td>
<td>commodity/tendering</td>
<td>Cloud</td>
</tr>
<tr>
<td>OpenPEX</td>
<td>bartering/double auction</td>
<td>Cloud</td>
</tr>
<tr>
<td>EERM</td>
<td>commodity/posted price/bartering</td>
<td>Cloud</td>
</tr>
</tbody>
</table>

D. Task Scheduling and Load-balancing Technique

A task is a piece of work to be completed within a specified time. Tasks in fixed set are statically assigned to processors, either at compile-time or at start-up (i.e. partitioning). The scheduling algorithms in distributed systems usually have the goals of spreading the load on processors and maximizing their utilization while minimizing the total task execution time.

Scheduling algorithms can be broadly categorized as centralized or decentralized, dynamic or static [5], or the hybrid policies in latest trend. A centralized load balancing approach can support larger system. Hadoop system takes the centralized scheduler architecture. In static load balancing, all information is known in advance and tasks are allocated according to the prior knowledge and will not be affected by the state of the system. Dynamic load-balancing mechanism has to allocate tasks to the processors dynamically as they arrive. Redistribution of tasks has to take place when some processors become overloaded [4].

In cloud computing, each application of users runs on a separate virtual machine. Every application is completely different and is independent and has no link between each other. Some applications require more CPU time to compute complex task, and some others may need more memory to store data, etc. So, to meet the different requirements of users, the virtual machines are available in various sizes i.e. medium, large and extra-large. Every individual unit use of resources (like CPU cost, memory cost, I/O cost, etc.) are measured.

E. Economic analysis of Task Scheduling Algorithms:

The cloud computing tasks are different from one another and it needs an optimal execution time. Cloud scheduler will assign the multiple users tasks to multiple virtual machines. A good scheduling algorithm should assign the virtual machines in an optimal way. So, scheduling is just a matching, task to machines. The optimal solution is obtained in a heuristic manner. Heuristic is applied as a optimal algorithm to obtain a good solution.

Here, we consider the following terms for our understanding:

<table>
<thead>
<tr>
<th>task</th>
<th>virtual machine</th>
<th>ti</th>
<th>mji</th>
</tr>
</thead>
<tbody>
<tr>
<td>the time when task ti arrives</td>
<td>ci</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the time when machine mji is available</td>
<td>aj</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the execution time for ti on mji</td>
<td>eij</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the time when the execution of ti is finished on mji</td>
<td>cij</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the maximum value of cij</td>
<td>makespan</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The goal of scheduling is to minimize the completion time (makespan) of all tasks for optimization of resource utilization and to reduce the resource cost.

3. EXISTING TASK SCHEDULING ALGORITHMS

The following task scheduling algorithms are currently prevalent in clouds.

Heuristics are applied to find an optimal solution to the scheduling. Heuristic can be both static and dynamic. Static heuristic is applied when the numbers of tasks to be completed are known in prior. Dynamic heuristic can be used when the task arrival is dynamic in nature.

3.1 Static Scheduling

Static strategies are performed under two assumptions:

i. Tasks arrive simultaneously at ci = 0.

ii. Machine available time aj is updated after each task is scheduled.

The static scheduling techniques are described below:

a) Opportunistic Load Balancing (OLB): schedules every task, in arbitrary order, to next available machine. Its implementation is quite easy, because it does not need extra calculation. The goal of OLB is simply keeping all machines as busy as possible.

b) Minimum Execution Time (MET) schedules every task, in arbitrary order, to the machine which has the minimum execution time for this task. MET is also very simple, giving the best machine to each task, but it ignores the availability of machines. MET jeopardizes the load balance across machines.

c) Minimum Completion Time(MCT) : schedules every task, in arbitrary order, to the machine which has the minimum completion time for this task. However, in this heuristic, not all tasks can be given the minimum execution time.

d) Min-min begins with the set T of all unscheduled tasks. Then, the matrix for minimum completion time for each task in set T is calculated. Task with overall minimum completion time is scheduled to its corresponding machine. Next, the scheduled task is removed from T. The process repeats until all tasks are scheduled.

e) Min-max is similar to Min-min heuristic. Min-max also begins with the set T of all unscheduled tasks, and then calculates the matrix for minimum completion time for each task in set T. Different from min-min, task with overall maximum completion time is selected and scheduled to its corresponding machine. Next, the scheduled task is removed from T. The process repeats until all tasks are scheduled.

f) GA (Genetic Algorithm) is a heuristic search for a near-optimal solution in large solution spaces [6]. The first step is randomly initializing a population of chromosomes (possible scheduling) for a given task. Each
chromosome has a fitness value (makespan) that results from the scheduling of tasks to machines within that chromosome. After the generation of the initial population, all chromosomes in the population are evaluated based on their fitness value, with a smaller makespan being a better mapping. Selection scheme probabilistically duplicates some chromosomes and deletes others, where better mappings have a higher probability of being duplicated in the next generation. The population size is constant in all generations. Next, the crossover operation selects a random pair of chromosomes and chooses a random point in the first chromosome. Crossover exchanges machine assignments between corresponding tasks. Mutation operation is performed after crossover. Mutation randomly selects a chromosome, then randomly selects a task within the chromosome, and randomly reassigns it to a new machine. After evaluating the new population, another iteration of GA starts, including selection, crossover, mutation and evaluation. Only when stopping criteria are met, the iteration will stop.

g) SA (Simulated Annealing) uses a procedure that probabilistically allows poorer solutions to be accepted to obtain a better search of the solution space. This probability is based on a system temperature that decreases for each iteration, which implies that a poorer solution is difficult to be accepted. The initial system temperature is the makespan of the initial scheduling, which is mutated in the same manner as the GA. The new makespan is evaluated at the end of each iteration. A worse makespan might be accepted based on a probability, so the SA finds poorer solutions than Min-min and GA.

h) Tabu search keeps track of the regions of the solution space which have already been searched so as not to repeat a search near these areas. A scheduling solution uses the same representation as a chromosome in the GA approach. To manipulate the current solution and to move through the solution space, a short hop is performed. The intuitive purpose of a short hop is to find the nearest local minimum solution within the solution space. When the short hop procedure ends, the final scheduling from the local solution space search is added to the tabu list. Next, a new random scheduling is generated, to perform a long hop to enter a new unsearched region of the solution space. After each successful long hop, the short hop procedure is repeated. After the stopping criterion is satisfied, the best scheduling from the tabu list is the final answer. \( A^* \) is a tree-based search heuristic beginning at a root node that is a null solution. As the tree grows, nodes represent partial scheduling (a subset of tasks is assigned to machines), and leaves represent final scheduling (all tasks are assigned to machines). The partial solution of a child node has one more task scheduled than the parent node. Each parent node can be replaced by its children. To keep execution time of the heuristic tractable, there is a pruning process to limit the maximum number of active nodes in the tree at any one time. If the tree is not pruned, this method is equivalent to an exhaustive search. This process continues until a leaf (complete scheduling) is reached.

3.2 Dynamic Scheduling

Dynamic heuristics are necessary when task set or machine set is not fixed. For example, not all tasks arrive simultaneously, or some machines go offline at intervals. The dynamic heuristics can be used in two fashions, on-line mode and batch mode. In the former mode, a task is scheduled to a machine as soon as it arrives. In the latter mode, tasks are firstly collected into a set that is examined for scheduling at prescheduled times.

On-line mode

In on-line heuristics, each task is scheduled only once, the scheduling result cannot be changed. On-line heuristic is suitable for the cases in which arrival rate is low [7].

a) OLB dynamic heuristic assigns a task to the machine that becomes ready next regardless of the execution time of the task on that machine.

b) MET dynamic heuristic assigns each task to the machine that performs that task’s computation in the least amount of execution time regardless of machine available time.

c) MCT dynamic heuristic assigns each task to the machine, which results in task’s earliest completion time. MCT heuristic is used as a benchmark for the on-line mode [7].

d) SA (Switching Algorithm) uses the MCT and MET heuristics in a cyclic fashion depending on the load distribution across the machines. MET can choose the best machine for tasks but might assign too many tasks to same machines, while MCT can balance the load, but might not assign tasks machines that have their minimum executing time. If the tasks are arriving in a random mix, it is possible to use the MET at the expense of load balance up to a given threshold and then use the MCT to smooth the load across the machines.

e) KPB (K-Percent Best) heuristic considers only a subset of machines while scheduling a task. The subset is formed by picking the k best machines based on the execution times for the task. A good value of k schedules a task to a machine only within a subset formed from computationally superior machines. The purpose is to avoid putting the current task onto a machine which might be more suitable for some yet-to-arrive tasks, so it leads to a shorter makespan as compared to the MCT.

Batch mode

In batch mode, tasks are scheduled only at regular intervals. This enables batch heuristics to know about the actual execution times of a larger number of tasks.

a) Min-min firstly updates the set of arrival tasks and the set of available machines, calculating the corresponding expected completion time for all ready tasks. Next, the task with the minimum earliest completion time is scheduled and then removed from the task set. Machine available time is updated, and the procedure continues until all tasks are scheduled.
b) Max-min heuristic differs from the Min-min heuristic where the task with the maximum earliest completion time is determined and then assigned to the corresponding machine. The Max-min performs better than the Min-min heuristic if the number of shorter tasks is larger than that of longer tasks.

c) Sufferage heuristic assigns a machine to a task that would suffer most if that particular machine was not assigned to it. In every scheduling event, a sufferage value is calculated, which is the difference between the first and the second earliest completion time. For task tk, if the best machine mj with the earliest completion time is available, tk is assigned to mj. Otherwise, the heuristic compares the sufferage value of tk and ti, the task already assigned to mj. If the sufferage value of tk is bigger, ti is unassigned and added back to the task set. Each task in set is considered only once.

### 3.3 Heuristic Scheduling

No general schedulers can fit for all cloud architectures. The following are the schedulers used for data-intensive distributed applications.

- **a) Hadoop MapReduce** is a popular computation framework for processing large-scaled data in mainstream public and private clouds, and it is considered as an indispensable cornerstone for cloud implementation. Hadoop is the most widespread MapReduce implementation for educational or production uses. It enables applications to work with thousands of nodes and petabytes of data.

- **b) FIFO scheduler** [8] applies first in first out heuristic. When a new job is submitted, scheduler puts it in the queue according to its arrival time. The earliest job on the waiting list is always executed first. The advantages are that the implementation is quite easy and that the overhead is minimal. However, throughput of FIFO scheduler is low, since tasks with long execution time can seize the machines.

- **c) Fair scheduler** [9] assigns equal share of resources to all jobs. When new jobs are submitted, tasks slots that free up are shared, so that each job gets roughly the same amount of CPU time. Fair scheduler supports job priorities as weights to determine the fraction of total compute time that each job should get. It also allows a cluster to be shared among a number of users. Each user is given a separate pool by default, so that everyone gets the same share of the cluster no matter how many jobs are submitted. Within each pool, fair sharing is used to share capacity between the running jobs. In addition, guaranteed minimum share is allowed. When a pool contains jobs, it gets at least its minimum share, but when the pool does not need its full guaranteed share, the excess is split among other running jobs.

- **d) Capacity scheduler** [10] allocates cluster capacity to multiple queues, each of which contains a fraction of capacity. Each job is submitted to a queue, all jobs submitted to the same queue will have access to the capacity allocated to the queue. Queues enforce limits on the percentage of resources allocated to a user at any given time, so no user monopolizes the resource. Queues optionally support job priorities. Within a queue, jobs with high priority will have access to resources preferentially. However, once a job is running, it will not be preempted for a higher priority job.

- **e) Delay scheduler** [11] addresses conflict between scheduling fairness and data locality. It temporarily relaxes fairness to improve locality by asking jobs to wait for a scheduling opportunity on a node with local data. When the job that should be scheduled next according to fairness cannot launch a local task, it waits for a short length of time, letting other jobs launch tasks instead. However, if a job has been skipped long enough, it is allowed to launch non-local tasks to avoid starvation. Delay scheduler is effective if most tasks are short compared to jobs and if there are many slots per node.

- **f) Dryad** Dryad [12] is a distributed execution engine for general data parallel applications, and it seems to be Microsoft’s programming framework, providing similar functionality as Hadoop. Dryad applies directed acyclic graph (DAG) to model applications.

- **g) Quincy** [13] scheduler tackles the conflict between locality and scheduling in Dryad framework. It represents the scheduling problem as an optimization problem. Min-cost flow makes a scheduling decision, matching tasks and nodes. The basic idea is killing some of the running tasks and then launching new tasks to place the cluster in the configuration returned by the flow solver.

### 3.4 Real-time scheduling for cloud computing

In real time cloud applications the cloud users and providers must have a strong service level agreement to ensure the timing of applications, and deadlines of applications. A real-time scheduler must ensure that processes meet deadlines, regardless of system load or makespan. Priority is applied to the scheduling of these periodic tasks with deadlines. Every task in priority scheduling is given a priority through some policy, so that scheduler assigns tasks to resources according to priorities. Based on the policy for assigning priority, real-time scheduling is classified into two types: fixed priority strategy and dynamic priority strategy.

### 1) Fixed priority strategies

The most influential algorithm for priority assignment is Rate Monotonic (RM) algorithm proposed by Liu [14]. In RM algorithm, in fixed priority scheduling the dispatcher will make sure that at any time the highest priority runnable task is actually running. So, if we have a task with a low priority running and a high priority task arrives. The low priority task will be suspended and the high priority task will start running. If while the high priority task is running a medium priority task arrives the dispatcher will leave it unprocessed and the
scheduling algorithms for real-time tasks, that is, in the environment. We then examine the particular applications play a significant role in cloud computing and then the scheduling when tasks arrive. Real-time scheduling when the complete set of tasks is known prior to execution, while dynamic heuristic performs the scheduling when tasks arrive. Real-time applications play a significant role in cloud environment. We then examine the particular scheduling algorithms for real-time tasks, that is, priority-based strategies. These strategies, already used in traditional real-time kernels, are not very capable of satisfying cloud systems requirements. New technologies, such as loadable real-time plug-ins and virtual machines, are introduced as promising solutions for real-time cloud schedulers.

### References

1. The NIST definition of Cloud computing, September 2011.