Job Scheduling in Cloud Environment based on Shuffled Frog Leaping Algorithm

S. Arjmand
Department of Computer Engineering, Yazd Branch, Islamic Azad University, Yazd, Iran

F. Adibnia*
Department of Computer Engineering, Yazd University, Yazd, Iran

*Corresponding Author: Fadib@yazd.ac.ir

Abstract

In this research, an optimized cloud task scheduling algorithm based on shuffled frog leaping algorithm (SFLA) is proposed. The classic methods of scheduling tasks in operating systems and cloud environment are FCFS, SJF, RR, etc. but researchers consider the scheduling algorithms as an optimization problem to achieve better and stable results. In this work, the proposed method is implemented by CloudSim and compared to FCFS and ACO algorithms to prove the high performance and efficiency. The results show that our algorithm is running in lower time and make span in comparison with FCFS as a classic algorithm and ACO as an optimization problem.

Keywords: Cloud computing; Job scheduling; Ant colony optimization; Shuffled frog leaping algorithm.
1) Introduction

The proliferation of cloud computing in computer science results in massive amount of researches which is recently done in this area specifically in cloud task scheduling problems. It is an emerging computing paradigm with foundations of grid computing, distributed systems, service oriented architecture and virtualization [1]. It includes many resources and requests, in an effort to share the resources as services on the Internet that typically divided into three levels of service offerings: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [2].

Cloud computing service providers, make the large-scale network servers form the pool of massive virtual resources. IaaS level is a hardware provider (server, storage and network), and associated software (operating systems, file system), and deliver a large amount of computational capacities to service remote users in a flexible and efficient way. In this level, resources have provisioned in the form of Virtual Machines (VMs) deployed within the cloud equipment’s consisting of: Data-center, physical resources and etc., for fulfilling the requests.

Task scheduling is a key process for IaaS; it means to assign the requests to resources in an efficient way, considering cloud characteristics. It takes VMs as scheduling units for mapping physical heterogeneous resources on tasks. Each VM is an abstract unit of computing and storage capacities in cloud. Although the task scheduling in heterogeneous environment such as cloud computing systems with dynamic characteristics is a NP-complete problem, so it has to be done immediately and automatically.

For this problem, plenty of methods proposed recently such as: Greedy (First-fit) [3], [4], [5] and Round-Robin (used by some cloud systems such as Eucalyptus [4]), queuing system (used in Open Nebula), advanced reservation and preemption scheduling [6], max-min [3] and etc. Some of them do not consider parameters such as maximum usage of physical resources, or resources’ load balancing and quick response.

To solve NP-complete problems, it almost is used evolutionary and heuristic algorithms, toward scheduling problem in distributed systems used some algorithms such as: Partial swarm optimization (PSO), Simulated Annealing, Tabu Search, Genetic Algorithm (GA) and etc. In this work, Shuffled Frog Leaping Algorithm (SFLA) has been implemented and examined, which outperformed the classic and modern optimization algorithms like FCFS and Ant Colony Optimization (ACO), to produce a proper scheduling with immediate response for cloud.

2) Related Work

Cloud task scheduling policies vary depending on the deployment model of the cloud. This section provides a brief review of some related work done in scheduling in cloud environment. Ye Hu, Johnny Wong, Gabriel Iszlai and Marin Litoiu [7] have proposed a new probability dependent priority algorithm to determine the minimum number of servers required to execute the jobs, considering jobs in two different classes. In [8] an optimized algorithm for task scheduling based on
Activity Based Costing, is presented, that selects a set of resources, to schedule the tasks, such that the profit is maximized. A heuristic method to schedule bag-of-tasks (tasks with short execution time and no dependencies) in a cloud is presented in [9] so that the number of virtual machines to execute all the tasks, within the budget, is minimum and at the same time speedup is maximum.

Hybrid cloud is a model which combines a private cloud and public cloud. That is, during peak load when there are not sufficient resources to execute a task in a private cloud, outsource the same to a public cloud provider and gets it done. An optimal scheduling policy based on linear programming; to outsource deadline constrained workloads in a hybrid cloud scenario is proposed in [10]. The scheduling policies used in Eucalyptus [4] are First Fit and Round Robin. In OpenNebula, Haizea can be used as the scheduler backend, which supports advance reservations in the form of a lease [11]. All these static algorithms have some disadvantages that they are not so flexible in dynamic cloud environments. Recently, many researches have been done in order to provide a dynamic behavior in cloud environments and meet the essential needs of cloud task schedulers. Meng Xu et al [12] and Qiyi and Tinglei [13] proposed some homogeneous methods which work optimized in many cloud environment and circumstances. Mell and Grance in 2011 at National Institute of Standards and Technology (NIST) defined a model of cloud system consisting three layer service model: IaaS, PaaS, SaaS with five characteristics and four deployment models which is being used in most researches after 2011 [14]. This model is shown in figure 1. Medhat et al proposed a method based on Ant Colony Optimization (ACO) as a heuristic to solve the cloud task scheduling problem. They proved that considering cloud task scheduling as an optimization problem is much more efficient than classic methods such as FCFS and RR [15].
3) Proposed Algorithm

3.1 SFLA

The Shuffled frog leaping algorithm is a meta-heuristic optimization method which is based on observing, imitating, and modeling the behavior of a group of frogs when searching for the location that has the maximum amount of available food [16]. Shuffled frog leaping algorithm, originally developed by Eusuff and Lansey in 2003, can be used to solve many complex optimization problems, which are nonlinear, non-differentiable, and multi-modal [17]. The most distinguished benefit of Shuffled frog leaping algorithm is its fast convergence speed [18]. Shuffled frog leaping algorithm is a population based random search algorithm inspired by nature memetics. In the Shuffled frog leaping algorithm, a population of possible solution defined by a group of frogs that is partitioned into several communities referred to as memeplexes. Each frog in the memeplexes is performing a local search. Within each memeplex, the individual frog’s behavior can be influenced by behaviors of other frogs, and it will evolve through a process of memetic evolution. After a certain number of memetics evolution steps, the memeplexes are forced to mix together and new memeplexes are formed through a shuffling process. The local search and the shuffling processes continue until convergence criteria are satisfied (eq.1).

The SFLA steps are as follows and the complete flowchart is shown in fig 1 and 2:

(1) The Shuffled frog leaping algorithm involves a population ‘P’ of possible solution, defined by a group of virtual frogs (n).
(2) Frogs are sorted in descending order according to their fitness and then partitioned into subsets called as memeplexes (m).
(3) Frogs i is expressed as Xi =(Xi1,Xi2,…..Xis ) where S represents number of variables.
(4) Within each memeplex, the frogs with worst and best fitness are identified as Xw and Xb.
(5) Frog with global best fitness is identified as Xg.
(6) The frog with worst fitness is improved according to the following equations.

\[
\begin{align*}
D_i &= \text{rand}(X_b - X_w) \\
X_{w_{\text{new}}} &= X_{w_{\text{old}}} + D_i (-D_{\text{max}} \leq D_i \leq D_{\text{max}})
\end{align*}
\]

Where rand is a random number in the range of [0,1]. Di is the frog leaping step size of the i-th frog and Dmax is the maximum step allowed change in a frog’s position. If the fitness value of new Xw is better than the current one, Xw will be accepted. If it isn’t improved, then the calculated (eq.1) and (eq.2) are repeated with Xb replaced by Xg. If no improvement becomes possible in the case, a new Xw will be generated randomly. Repeat the update operation for a specific number of iterations.
Fig. 2. SFLA Flowchart

Fig. 3. Local Search Flowchart
After a predefined number of mimetic evolutionary steps within each memeplex, the solutions of evolved memeplexes are replaced into new population. This is called the shuffling process. The shuffling process promotes a global information exchange among the frogs. Then, the population is sorted in order of decreasing performance value and updates the population best frog’s position, repartition the frog group into memeplexes, and progress the evolution within each memeplex until the conversion criteria are satisfied [16].

3.2 Cloud SIM

Simulation is an essential part of computer science because sometimes it’s not possible to evaluate an algorithm on a real system or in this case internet. For simulation we need a special toolkit named CloudSim. It is basically a Library for Simulation of Cloud Computing Scenarios. It has some features such as it support for modeling and simulation of large scale Cloud Computing infrastructure, including data centers on a single physical computing node. It provides basic classes for describing data centers, virtual machines, applications, users, computational resources, and policies. CloudSim supports VM Scheduling at two levels: First, at the host level where it is possible to specify how much of the overall processing power of each core in a host will be assigned at each VM. And the second, at the VM level, where the VMs assign specific amount of the available processing power to the individual task units that are hosted within its execution engine [19]. Most important objects which is supported in CloudSim are Datacenter, Host, Virtual Machine, CloudLet, VmScheduler, CloudLetScheduler.

3.3 Task Scheduling Based On SFLA

The SFLA algorithm has been customized and adopted in order to schedule cloud based tasks. The problem is defined as follows: set of CloudLets CLs={CL1, CL2, …, CLc} which c is the number of Tasks that must be run on cloud environment and set of VirtualMachines VMs={VM1, VM2, …, VMv} which v is the total number of processing units or virtual machines. The purpose of scheduling algorithms is achieving the best solution which has minimum make span in this particular case. Make span is the overall execution time of all tasks on provided processing nodes or VMs. In our case, each frog is represents a solution which is some assignments between CLs and VMs and could be shown as groups. For an instance, a frog is {(CLa,VMx),( CLb,VMy),…} that tell us CLa is run on VMx and so forth.

In order to score the frogs, a cost function or fitness function is used and it’s unexceptionally the same stage in all optimization algorithms. In this problem, the eq.3 [15] is used in SFLA and ACO algorithms to make a true comparison.

\[
\text{Fitness}_{ij} = \frac{\text{TaskLength}_{ij}}{PE_{ij} \times VMmips_{ij}} + \frac{\text{InputFileSize}_{ij}}{VMbw_{ij}}
\]
In the above equation, Fitness \( ij \) means the cost of running task \( i \) on VM \( j \) which the task features are task length and file size and the properties of VM are number of processing elements, power of each PE and band width of VM. By this function, we can compute the fitness of each frog, sort and split them to make some evolutionary steps. The complete pseudo code of proposed method is shown in Fig.4.

**Input:** number of CloudLets and VMs and their specifications

**Output:** best optimized solution

Generate random population of \( P \) solutions (frogs);

While Iteration is needed

For each individual, calculate fitness Eq.3

Sort the population \( P \) in descending order of their fitness;

Divide \( P \) into \( m \) memeplexes;

For each generation

For each memeplex

Determine the best and worst frogs;

Improve the worst frog position using Eqs. (1), (2);

End;

Combine the evolved memeplexes;

Sort the population \( P \) in descending order of their fitness;

Check if termination=true then exit;

End;

End;

End;

Fig. 4. Pseudo code of proposed scheduling algorithm

### 3.4 Experimental Results

We implemented a cloud based job scheduling scenario by Java programming language and Cloud SIM library which three polices are used to schedule tasks. All implemented algorithms is written in Eclipse programming IDE and tested on a PC with 2GB RAM, CORE I5 CPU and Windows 7 as an operating system. The table 1 shows the shared parameters which is used in FCFS, ACO and SFLA. The ranged entries show that they are variable in each execution of program and randomly selected in the execution time in order to carry out a real experiment.
Table 1. CloudSim Parameter Specifications

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudLets</td>
<td></td>
</tr>
<tr>
<td>Length of tasks</td>
<td>2000 – 20000</td>
</tr>
<tr>
<td>Total number of tasks</td>
<td>50 – 500</td>
</tr>
<tr>
<td>Virtual Machines</td>
<td></td>
</tr>
<tr>
<td>Total number of VMs</td>
<td>10</td>
</tr>
<tr>
<td>Mips</td>
<td>500 – 2000</td>
</tr>
<tr>
<td>VM memory</td>
<td>512</td>
</tr>
<tr>
<td>Band Width</td>
<td>500 – 1000</td>
</tr>
<tr>
<td>Cloudlet Scheduler</td>
<td>Space shared</td>
</tr>
<tr>
<td>Processing elements</td>
<td>1 – 4</td>
</tr>
<tr>
<td>Data Centers</td>
<td></td>
</tr>
<tr>
<td>Total number of Data Centers</td>
<td>10</td>
</tr>
<tr>
<td>Number of hosts</td>
<td>6</td>
</tr>
<tr>
<td>VM scheduler</td>
<td>Time shared</td>
</tr>
</tbody>
</table>

One of the most prominent factors to compare the efficiency of algorithms is the running time. Here we simulate a task scheduling algorithm on the same cloudlet set and virtual machines with three different policies and the results are shown in table 2. In the following table we can normally ignore the FCFS column because it is not an optimization algorithm and it is usual to get such a small running time.

Table 2. Running time comparison for 50 cloudlet

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>SFL</th>
<th>ACO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFS</td>
<td>1</td>
<td>334</td>
<td>1850</td>
</tr>
<tr>
<td>ms</td>
<td>ms</td>
<td>ms</td>
<td>ms</td>
</tr>
</tbody>
</table>

The SFLA is running almost five times faster than ACO. Apart from that, this measured times are recorded for 100 iterations of related algorithms so the approaching to the answer is different and comparable which the fig 5 and 6 show the convergence of ACO and SFLA. ACO is reached to the best answer after 33 iterations but SFLA is reached to the same point after 22 iterations so it is clear that the convergence speed of SFLA is better than that of ACO at least in this particular case.
Furthermore, the other important criteria to make some contrasts between scheduling algorithms is make span which is the estimated running time of certain solution on cloud. The Fig 7 shows the make span of three examined algorithms on different number of tasks.

To make a reliable comparison, we run each algorithm ten times and average the output make spans. Needless to say that ACO and SFLA make different results in each running of program because of its optimization behavior but FCFS is a linear algorithm and makes the same result for a particular set of inputs.
4) Conclusion

In this paper the SFLA as a cloud based job scheduling algorithm presented. In order to make a dependable comparison, the ACO and FCFS algorithms were implemented and three criteria defined and compared (make span, running time and convergence speed). Finally, we exhaustively extract the results and figured out that SFLA outperforms the ACO and FCFS. In the future works, other meta-heuristic methods will be examined as a cloud based job scheduling algorithm.
References


