Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 151 (2019) 1078–1083

Procedia Computer Science

www.elsevier.com/locate/procedia

International Workshop on Peer-to-Peer Architectures, Networks, and Systems (P2PAN)
April 29 – May 2, 2019, Leuven, Belgium

Cuckoo-inspired Job Scheduling Algorithm for Cloud Computing

Ebtesam Aloboud[a,*], Heba Kurdi[b]

[a]*Computer Science Department, Al Imam Mohammad Ibn Saud Islamic University, Riyadh, SA*
[b]*Computer Science Department, King Saud University, Riyadh, SA*

Abstract

Cloud computing offers a pool of virtual resources to users on demand. Among the main challenge facing cloud computing is job scheduling which is an NP-hard problem. Usually, scheduling should consider job priority, where high-priority jobs should be immediately allocated required resources, whereas low-priority jobs can wait. In this paper, we propose a scheduling algorithm that imitates the parasitic behaviour of the cuckoo bird. Cuckoos reproduce by exploiting the nests of other birds with eggs similar to their own. We have evaluated the proposed cuckoo scheduling algorithm under varying number of nodes and percentage of high priority jobs. The experimental results demonstrate the superiority of our proposed algorithm over the benchmarks in terms of average CPU utilization and average turnaround time for each type of job.

Keywords: Bio-inspired; Cuckoo bird; Job Priority Scheduling; Cloud Computing

1. Introduction

Cloud Computing is emerging as an infrastructure that decreases the need for physical hardware. It offers advantages over traditional Information Technology (IT) infrastructure in scalability, elasticity, and utilization. Data centres enable cloud consumers to access distributed resources, through virtualization, as much as they want and wherever they need them [1]. One cloud computing deployment model that is recognizable to consumers is the public cloud. Its services are provided in a virtualized environment using shared infrastructure for open use by the public over the internet, such as Amazon Elastic Compute Cloud (EC2) and Windows Azure [6][7].

* Corresponding author. Tel.: +966-11-25-97-548.
E-mail address: emoboud@imamu.edu.sa

10.1016/j.procs.2019.04.153

Scheduling is one of the most important challenges in cloud computing because of the enormous number of virtual machine (VM) requests and the infinite pool of physical resources. Job scheduling is considered as an NP-hard problem that uses heuristic algorithms to approximate solution. Scheduling in the cloud can occur in a static mode, where all jobs are submitted at once and resource allocation is pre-computed before system execution [11][12]. In a dynamic mode, the arrival of jobs is unpredictable and resource allocation is completed in real-time. Job prioritisation might complicate the scheduling process as some jobs can wait, while others cannot [11]. Moreover, High-priority jobs pre-empt low-priority jobs, which might delay completion of the job [3][11].

Although, many researchers have contributed to job scheduling in cloud, there is still a scope for improvement. In [3], the low-priority job is called a best-effort lease (BE), and must be scheduled as a whole or not at all. If the schedule cannot map all the VMs for a BE job, it reports that there is no resource available. In [4], Nathani proposed introducing deadlines for best-effort jobs and offered two algorithms to support best-effort job pre-emption: in swapping, two consecutive jobs are swapped if the first job has requested fewer resources than the second one; in backfilling, the idle resources are filled by low-priority jobs, prioritized by their deadlines. In [10] a modified version of Ant Colony Optimization technique that uses ants to keep track of overloaded and under-loaded nodes. In [13], have proposed a hybrid Cuckoo and Particle Swarm Optimisation algorithm to generate an optimal task schedule to satisfy a minimum execution time, as well as, resources utilization.

The primary aim of this paper is to design a bio-inspired algorithm that emulates the cuckoo bird in scheduling its eggs on another bird's nest to minimize turnaround time and maximize processor utilisation. The proposed algorithm has been tested on a public cloud simulator and benchmarked against existing algorithm. The remainder of this paper is organized as follows: In Section 2, we present cuckoo behaviour. Section 3 explains the development of the algorithm based cuckoo behaviour. In Section 4, we present the methodology used to evaluate the algorithm. Experimental results are presented in Section 5, and the conclusions are given in Section 6.

2. Cuckoo Behaviour Inspiration

In nature, birds usually exhibit parental care behaviours such as preparing a nest, provisioning food, and defending offspring. Many kinds of bird utilize parasitic brooding by exploiting resources allocated by other birds [2]. The majority of cuckoo birds are brood parasites; they reproduce by exploiting the nests of other birds for their own eggs. When the female cuckoo is ready to lay its egg, it searches for a different host nest for each of its egg. This strategy is used because some host species may discover the cuckoo egg in their nest and either abandon their nest or throw this egg away. Additionally, the cuckoo needs its hatchlings to get all the resources of the other bird. To ensure that at least some of its egg avoid detecting by the host bird, the cuckoo lays its eggs in nests of birds whose eggs closely resemble those of the cuckoo [2][5].

3. From Inspiration to Algorithm

Cuckoo birds do not have any nests of their own, relying instead completely on brood parasitism behaviour. In our theory, we mimic the way that the cuckoo bird lays its own eggs in different nests without having its own nest by creating a scheduling algorithm that fragments low-priority job into sub-jobs and allocates each sub-job to different hosts rather than assigning a specific host to all job segments in order to utilize other resources when possible. We fragment our cuckoo job into sub-jobs in such a way that if there is a high-priority job that would pre-empt our job as a whole, it will not pre-empt all the sub-jobs. The cuckoo bird searches for the first suitable match for its egg. In our approach, the algorithm searches for the first suitable host that achieves selected threshold.

In our algorithm, a low-priority pre-emptive job with no deadline is a cuckoo job. Each sub-job is a cuckoo egg. Physical host nodes are the host nests. The host bird is represented by a high-priority job that has an exact start time, that needs to be schedule at once, and that can pre-empt a low-priority sub-job if it is found. The host bird is the primary owner of the nest and has the right to remove the cuckoo’s egg once the egg (sub-job) is discovered.

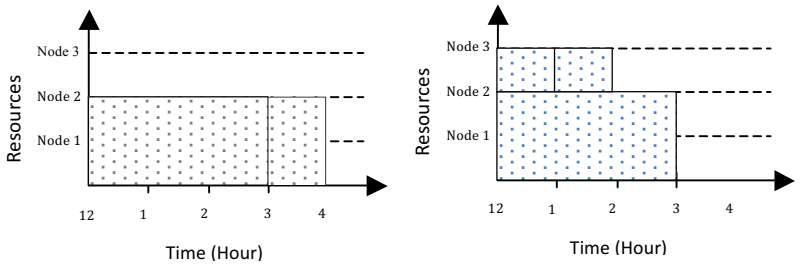
Table 1. Mapping between cuckoo in nature and cuckoo scheduling algorithm

Cuckoo in nature	Cuckoo in scheduling
Cuckoo's brood parasitism behaviour	Cuckoo scheduling algorithm
Cuckoo bird	Low-priority job
Host bird	High-priority job
Cuckoo's eggs	Cuckoo sub-jobs
Host nest	Physical node (resource)

3.1. System Design

Our system consists of the following main components:

Jobs: A job is a consumer's request from the provider, where the former requests a single VM or a group of VM from the latter based on the job specifications. Job specifications describe the request in terms of the necessary CPU power and memory. These jobs can be divided into two categories: high-priority jobs and cuckoo jobs. High-priority jobs have the right to pre-empt low-priority jobs. They have an exact time to start. If the scheduler cannot find resources to start this job, it is rejected. Cuckoo jobs are pre-emptive low-priority jobs with no deadline. Cuckoo jobs are split into sub-jobs, where each sub-job contains a request for a single VM, allowing the job to be allocated to different physical nodes at different times. Our model uses a fragmentation method to ensure utilization. The graphs in Fig. 1 (a) and (b) provide a visual comparison between non-fragmented low-priority jobs that has been implemented in [3] and cuckoo jobs. We assume that the total number of physical resources in the system is three nodes and that there are two different jobs: the first one requests two nodes for three hours and the second one requests two nodes for one hour. In Fig. 1 (a), when the second-non-fragmented job arrives, only one node will be available, so the second job will wait for the previous job to finish before starting. In Fig. 1 (b), the job is divided into two sub-jobs; one sub-job will work for one hour and then the other sub-job will work for one hour, which means the second job will finish before the first job is finished.



a) Non-Fragmented Low Priority Job b) Cuckoo Job

Fig.1 comparison between Non-Fragmented Low priority and Cuckoo job

Host Node: Host nodes are physical resources that host multiple VMs for different consumers. There are number of hosts in our system environment. Each of them has different CPU and memory specifications. The cuckoo scheduling algorithm schedules the jobs on the physical resources that fit the job requirements. The system tracks the current capacity of each host node.

Queue: once the system receives a cuckoo job request, the cuckoo is divided into sub-jobs and pushed into a queue. Each of these sub-jobs is removed from the queue and sent to the scheduler. If a sub-job is pre-empted, then it is returned to the queue.

Cuckoo Scheduling Manager: Cuckoo algorithm is used to search for a suitable host for the job, focusing on reducing waiting time. The first step is to determine the set of resources that satisfy the minimum requirements of users. In this step, it filters out resources that do not satisfy the minimal job requirements. In the second step, when a list of available hosts that fit the job is returned, a host for the job is randomly selected. Then, the manager calls a fitness function to score a given host using two parameters whose have effect on low-priority job pre-emption: (1) tight_fit_capacity, which describes how well the job fits the new host taking into consideration the number of future

jobs scheduled in that physical node and (2) slot_duration, which describes how long the job will work without suspension (number of job pre-emptions that might occur). Fitness function is defined by the following formula:

$$\text{Fitness\_function} = \text{tight\_fit\_capacity} + \text{slot\_duration}$$

If the host score is bigger than or equal to a certain threshold, then the search will stop, and the request will be allocated to the host. The aim is to find the best possible hosts in the quickest way to reduce allocation waiting time.

Algorithm 1. Cuckoo Host Selection Policy

```
Begin
        Host_List = Get a list of hosts that might fit the Job
        Selected_Host = Retrieve one host from Host_List
        While (Host_List is not empty AND Fitness_Function(Selected_Host) < threshold)
                Host = Retrieve one Host and remove it from Host_List
                IF (Fitness_Function(Host) > Fitness_Function(Selected_Host)) THEN
                        Selected_Host = Host;
                End
        End while
        Allocate the job with Selected_Host
End
```

4. Evaluation Methodology

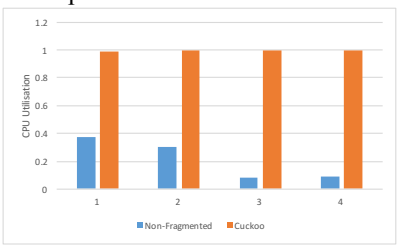
This proposed algorithm has been implemented using Python and tested using an empirical evaluation framework Haizea in simulation mode, which caused a modification in the Haizea scheduler [9]. It is compared to the non-fragmented low-priority job which is proposed in [3] and already implemented in Haizea. This scheduler has been used in simulation mode using clusters consisting of three nodes with specifications including HP ProLiant DL380p Gen8 E5-2620 1P 32GB-R 900GB 460W and Centos as an operating system. The workload is constructed using the Nordugird.org 14 dataset, which is formatted as a Grid Workload [8]. The job trace contains 781370 jobs with 1024 nodes. From this job trace, five hundred consecutive jobs were used to generate Haizea XML trace files, including other data, such as arrival time, duration, and the number of requested nodes. These workloads are used with the following configurations: using the AR-pre-empts-everything policy, which means that a high-priority job can pre-empt any non-fragmented low-priority or cuckoo job to take its slot. The experimental evaluation framework was:

- Determining the parameters that affect the behaviour of the algorithms: number of nodes requested by each job; inter-arrival time (IT), which is the time between two consecutive jobs; and the percentage of high-priority jobs.
- Varying the experimental variables to test the algorithms under different conditions (high and low demands), which are: number of requested nodes: values were selected from {50, 150, 250, 350}; inter-arrival time (IT): values were selected from {32, 128, 512} minutes; percentage of high-priority jobs compared to the percentage of cuckoo jobs in our proposed approach and to the percentage of non-fragmented low-priority jobs: values were selected from {10, 20, 30}.
- Identifying the performance measures to analyse algorithms, which are: the average CPU utilization, which the CPU in cloud should be maximised; average turnaround time, the time of submission minus the time of completion should be minimised.
- Each of these experiments uses the following resources: 360 physical nodes, each having one CPU and 1024 MB memory, which are considered the total available resources on the cloud-provider side.

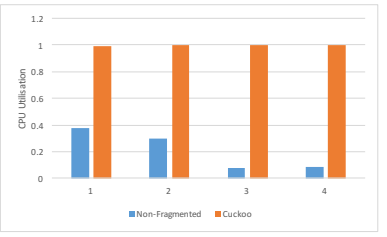
5. Result and Discussion

Fig.2, Fig.3, and Fig.4 compare the CPU utilization between a non-fragmented low-priority job (NFL) and cuckoo job (CJ) when the percentage of high-priority jobs (HPJ) equals 10%, 20%, and 30%, respectively, for Inter-arrival times of 32, 128, and 512 minutes. In Fig.2 and Fig.3, the HPJ is less than 20%, and when the number of requested nodes is 50, the CPU utilization for CJ is approximately 61% higher than that of NFL. For 150, the number of requested nodes increases by around 70%. For 250 requested nodes, it increases by approximately 91%. And for 350, it increases by around 90%. This case showed that with more CJ and its fragmented sub-jobs, we get better utilization. Similarly, Fig.4 represents the result of CPU utilization when the percentage of HPJ is 30% for the same IT, which shows that the CPU Utilization in CJ is higher than that of NFL by 56%, 66%, 90%, and 88% for

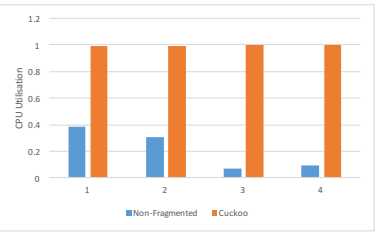
50, 150, 250, and 350 requested nodes, respectively. As seen above, the CPU utilization is better and reaches its peak with 90% at 250 nodes and then decreases to 80% at 310 nodes due to the intensive number of requests, which indicates that there are no available resources even for a fragmented small one. These figures prove that CJ sub-jobs can improve the CPU utilization.



(a) IT = 32

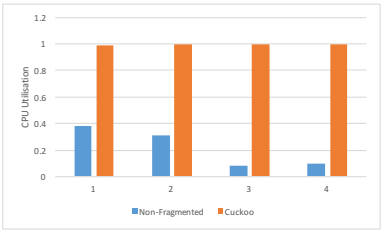


(b) IT = 128

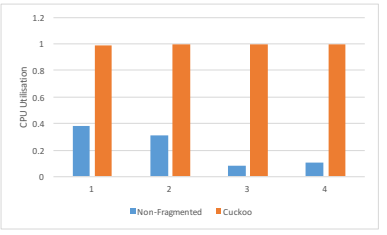


(c) IT = 512

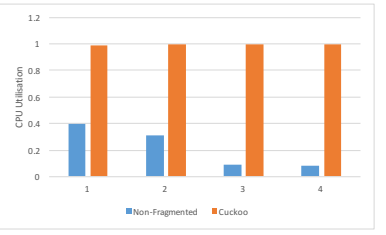
Fig.2 CPU Utilization when HPJ=20%



(a) IT = 32

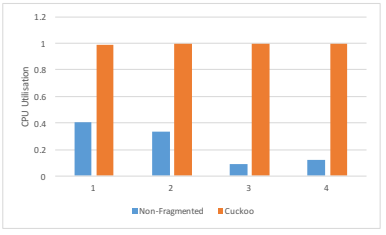


(b) IT = 128

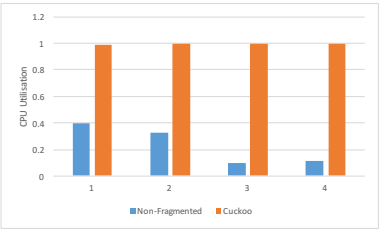


(c) IT = 512

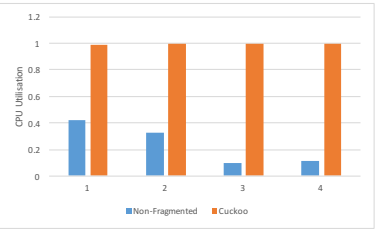
Fig.3 CPU Utilization when HPJ=20%



(a) IT = 32



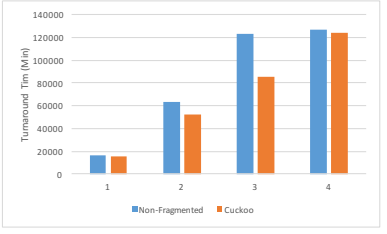
(b) IT = 128



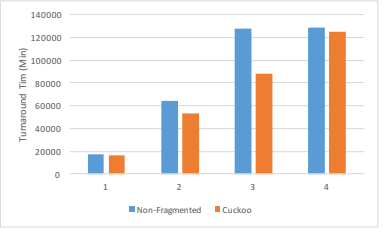
(c) IT = 512

Fig.4 CPU Utilization when HPJ=30%

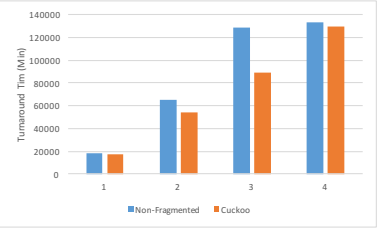
Fig.5, Fig.6, and Fig.7 compare the turnaround time between CJ and NFL. These results show that the turnaround time in CJ is less than the NFL. The NFL is slower than our approach for inter-arrival times of 32, 128 and, 512 minutes and 50, 150, 250, and 350 requested nodes, respectively, when the HPJ equals 10% by approximately 501 min, 10806 min, 38769 min, and 3696 min. The differences between the two approaches are reduced when the HPJ equals 20% and 30%. These reductions are reasonable as the number of HPJs are increased causing a delay in LPJs turnaround time. The better CJ turnaround time might encourage the consumer to request CJ to save money and get its job completed in a reasonable time.



(a) IT = 32

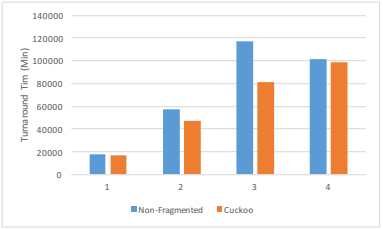


(b) IT = 128

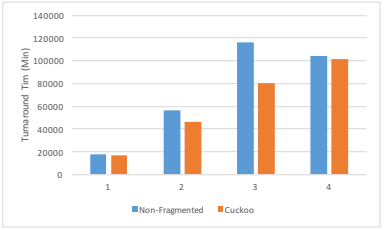


(c) IT = 512

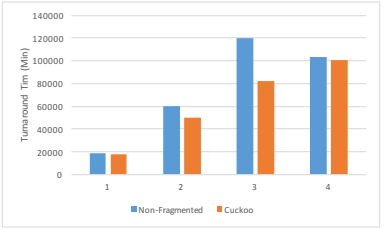
Fig.5 Turnaround Time when HPJ=10%



(a) IT = 32

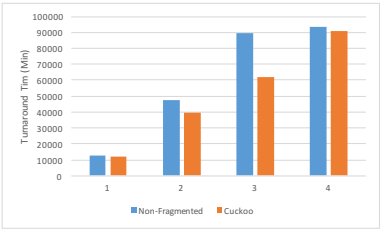


(b) IT = 128

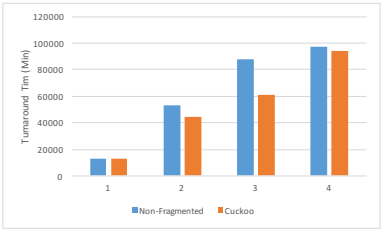


(c) IT = 512

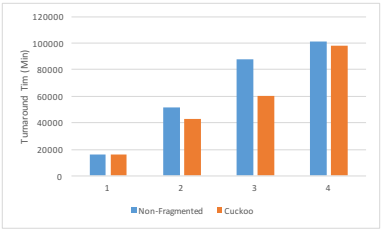
Fig.6 Turnaround Time when HPJ=20%



(a) IT = 32



(b) IT = 128



(c) IT = 512

Fig.7 Turnaround Time when HPJ=30%

6. Conclusion

The previous section explained how cuckoo low-priority jobs have outperformed non-fragmented low-priority jobs in two performance measurements, CPU utilization and turnaround time. The cuckoo algorithm used minimal time to execute the task due to the fragmentation of the job, which reduces the waiting time for resources to become available and increases their utilization time.

References

[1] Mell P, Grance T. The NIST definition of cloud computing.

[2] Soler M. Brood Parasitism in Birds: A Coevolutionary Point of View. In Avian Brood Parasitism 2017 (pp. 1-19). Springer, Cham.

[3] Sotomayor B, Montero RS, Llorente IM, Foster I. Resource leasing and the art of suspending virtual machines. In2009 11th IEEE International Conference on High Performance Computing and Communications 2009 Jun 25 (pp. 59-68). IEEE.

[4] Amit Nathani, Sanjay Chaudhary, Gaurav Somani, “Policy based resource allocation in IaaS cloud”, Future Generation Computer Systems 28, 94–103, Elsevier, June 2011.

[5] Kurdi H, Ezzat F, Altoaimy L, Ahmed SH, Youcef-Toumi K. MultiCuckoo: Multi-Cloud Service Composition Using a Cuckoo-Inspired Algorithm for the Internet of Things Applications. IEEE Access. 2018;6:56737-49.

[6] Amazon EC2, 2019, [online] https://aws.amazon.com/ec2/

[7] Microsoft Azure, 2019, [online] https://azure.microsoft.com/en-us/

[8] Nordugird, 2019, [Online]. http://www.nordugrid.org

[9] Haizea Scheduler, 2019, [online] http://haizea.cs.uchicago.edu

[10] Nishant K, Sharma P, Krishna V, Gupta C, Singh KP, Rastogi R. Load balancing of nodes in cloud using ant colony optimization. In 2012 UKSim 14th International Conference on Computer Modelling and Simulation 2012 Mar 28 (pp. 3-8). IEEE.

[11] Guddeti RM, Buyya R. A hybrid bio-inspired algorithm for scheduling and resource management in cloud environment. IEEE Transactions on Services Computing. 2017 Mar 8.

[12] Sotomayor B, Keahey K, Foster I. Combining batch execution and leasing using virtual machines. InProceedings of the 17th international symposium on High performance distributed computing 2008 Jun 23 (pp. 87-96). ACM.

[13] Al-maamari A, Omara FA. Task scheduling using hybrid algorithm in cloud computing environments. Journal of Computer Engineering (IOSR-JCE). 2015;17(3):96-106.