Research Article **ISSN: 2395 – 5597**

International Journal of Engineering and Robot Technology

Journal home page: www.ijerobot.com

NOVEL SCHEDULING ALGORITHMS FOR EFFICIENT DEPLOYMENT OF MAP REDUCE APPLICATIONS IN HETEROGENEOUS COMPUTING ENVIRONMENTS

J. Indumathi[*1]

[1*]Er. Perumal Manimekalai College of Engineering, Nallagana kothapalli, Near Koneripalli post, Hosur, Krishnagiri, Tamil Nadu, India.

ABSTRACT

Cloud computing has become increasingly popular model for delivering applications hosted in large data centers as subscription oriented services. Hadoop is a popular system supporting the Map Reduce function, which plays a crucial role in cloud computing. The resources required for executing jobs in a large data center vary according to the job type. In Hadoop, jobs are scheduled by default on a first-come-first-served basis, which may unbalance resource utilization. This paper proposes a job scheduler called the *job allocation scheduler* (JAS), designed to balance resource utilization. For various job workloads, the JAS categorizes jobs and then assigns tasks to a *CPU-bound queue* or an *I/O-bound queue*. However, the JAS exhibited a locality problem, which was addressed by developing a modified JAS called the *job allocation scheduler with locality* (JASL). The JASL improved the use of nodes and the performance of Hadoop in heterogeneous computing environments. Finally, two parameters were added to the JASL to detect inaccurate slot settings and create a dynamic job allocation scheduler with locality (DJASL). The DJASL exhibited superior performance than did the JAS, and data locality similar to that of the JASL.

KEYWORDS

Hadoop, Heterogeneous environments, Heterogeneous workloads, Map Reduce and Scheduling.

Author for Correspondence:

Indumathi J,

Er. Perumal Manimekalai College of Engineering,

Nallaganakothapalli, Near Koneripalli post,

Hosur, Krishnagiri, Tamil Nadu, India.

Email: indu_jsk_1987@yahoo.co.in

INTRODUCTION

The scale and maturity of the Internet has recently increased dramatically, providing excellent opportunities for enterprises to conduct business at a global level with minimum investment. The Internet enables enterprises to rapidly collect considerable amounts of business data. Enterprises must be able to process data promptly. Similar requirements can be observed in scientific and Big Data applications.

Therefore, promptly processing large data volumes in parallel has become increasingly imperative. Cloud computing has emerged as a new paradigm that supports enterprises with low-cost computing infrastructure on a pay-as-you-go basis. In cloud computing, the Map Reduce framework designed for parallelizing large data sets and splitting them into thousands of processing nodes in a cluster is a crucial concept. Hadoop which implements the MapReduce programming framework, is an open-source distributed system used by numerous enterprises, including Yahoo and Facebook, for processing large data sets. Hadoop is a server-client architecture system that uses the master-and-slave concept. The master node, called *Job Tracker*, manages multiple slave nodes, called *ask Tracker*s, to process tasks assigned by the *Job Tracker*[1].

MODULES

- Individual Performance of Each Workload.
- Performance analysis of the Job Allocation Scheduler and Job Allocation Scheduler Locality.
- Performance of the Dynamic Job Allocation and Scheduler Locality.

Individual Performance of Each Workload

The individual performance of each jobs, and each job setup comprised nearly 10 GB of data. The average execution time of the DJASL was compared with that of the default Hadoop algorithm in Environment 1 the results revealed that the sorting type jobs registered a higher execution time than the other jobs did, and that the join type jobs exhibited a shorter execution time. However, as shown in when multiple data were batch processed, the execution time did not increase multiples in continuation of the experiment. For example, if we have double data size of workloads, but the execution time will increase less than two times.

Performance analysis of the Job Allocation Scheduler and Job Allocation Scheduler Locality

In some of the ten requests, the performance of the JAS algorithm was not superior to those of Hadoop and DMR because the JAS algorithm sets slots inappropriately. Therefore, the resource utilizations of some *Task Tracker*s became overloaded, and some tasks could not be executed until resources were released. Hence, the execution times of these tasks increased, causing the performance of the JAS algorithm to decrease compared with those of Hadoop and DMR.

Performance of the Dynamic Job Allocation and Scheduler Locality

The *Job Tracker* occasionally inaccurately sets the slots when the JASL algorithm is applied, potentially reducing the performance. Hence, the DJASL algorithm includes two parameters, namely *CPU count* and *IO count*, which are used to ensure accurate slot settings. The *Job Tracker* resets slots according to threshold values, and differences in the threshold values cause performance results to vary. If a threshold value is too high (i.e., slots are set incorrectly when the DJASL is applied), the *Job Tracker* must wait for a long period to reset the slots. By contrast, if a threshold value is too low, the *Job Tracker* must reset slots frequently[2].

SYSTEM DESIGN[3-6]

Hadoop Default Scheduler

Hadoop supports the Map Reduce programming model originally proposed by Google [9], and it is a convenient approach for developing applications (e.g., parallel computation, job distribution, and fault tolerance). Map Reduce comprises two phases. The first phase is the map phase, which is based on a divide-and-conquer strategy. In the divide step, input data are split into several data blocks, the size of which can be set by the user, and are then paralleled by a map task. The second phase is the reduce phase. A map task is executed to generate output data as intermediate data after the map phase is complete, and these intermediate data are then received and the final result is produced. By default, Hadoop executes scheduling tasks on an FCFS basis, and its execution consists of the following steps:

Step 1

Job submission

When a client submits a Map Reduce job to a *Job Tracker*, the *Job Tracker* adds the job to the *Job Queue.*

Step 2

Job initialization

The *Job Tracker* initializes the job in the *Job Queue* by the *Job Tracker* by splitting it into numerous tasks; the *Job Tracker* then records the data locations of the tasks.

Step 3

Task assignment

When a *Task Tracker* periodically (every 3 seconds by default) sends a *Heartbeat* to a *Job Tracker*, the *Job Tracker* obtains information on the current state of the *Task Tracker* to determine whether it has available slots.

Job Workloads

Proposed that jobs can be classified according to the resources used; some jobs require substantial amount of computational resources, whereas other jobs require numerous I/O resources. In this study, jobs were classified into two categories according to their corresponding workload:

1) CPU-bound jobs and 2) I/O-bound jobs.

Hadoop Problem

As mentioned, Hadoop executes job scheduling tasks on an FCFS basis by default. However, this policy can cause several problems, including imbalanced resource allocation. Consider a situation involving numerous submitted jobs that are split into numerous tasks and assigned to *Task Tracker*s. Executing some of these tasks may require only CPU or I/O resources.

Because the default job scheduler in Hadoop does not balance resource utilization, some tasks in the *Task Tracker* cannot be completed until resources used to execute other tasks are released. Because some tasks must wait for resources to be released, the task execution time is prolonged, leading to poor performance.

Dynamic Map-Reduce Scheduler

To address the imbalanced resource allocation problem of the default scheduler in Hadoop, as described in Section 2.3, proposed a balanced resource utilization algorithm (DMR) for balancing CPU- and I/O-bound jobs. They proposed a classification-based triple-queue scheduler to determine the category of one job and then parallelize various job types and thus balance the resources of *Job Tracker*s by using CPU- and I/O-bound queues.

It then assigns two CPU-bound job tasks, *J*1*t*1 and *J*1*t*2, and two I/O-bound job tasks, *J*2*t*1 and *J*2*t*2, to *Task Tracker*1.

*Task Tracker*3 can execute one CPU-bound job and three I/O-bound jobs simultaneously (i.e., *Task Tracker*3 has three CPU slots and one I/O slot). Nevertheless, according to the DMR approach, each *Task Tracker* has two CPU slots and two I/O slots (implying a total of four slots). After receiving jobs from clients, the *Job Tracker* assigns the tasks to a *Task Tracker*. Each *Task Tracker* contains two CPU-bound tasks and two I/O-bound tasks. Therefore, *Task Tracker*1 has one I/O-bound task that must wait for the I/O resources to be released, resulting in its I/O capacity becoming overloaded. *Task Tracker*2 has one CPU slot that must wait for CPU resources to be released; therefore, the CPU capacity of *Task Tracker*2 becomes overloaded. Finally, *Task Tracker*3 has one CPU slot that must wait for CPU resources to be released; therefore, the CPU capacity of *Task Tracker*3 becomes overloaded. Furthermore, *Task Tracker*3 includes one idle I/O slot, indicating that its I/O resources are not effectively used. According to this example, the DMR may exhibit poor performance in a heterogeneous environment because of its inefficient resource utilization. Therefore, resource allocation is a critical concern in heterogeneous computing environments involving varying job workloads.

RESULTS

The experimental results can be classified into three themes presented in three sections: 1) Section 4.2.1 presents the individual performance of each job and indicates the effect of various data sizes; (2) Section 4.2.2 shows that the JAS algorithm improves the overall performance of the Hadoop system and that the JASL algorithm improves the data locality of the JAS; and 3) Section 4.2.3 indicates that the proposed DJASL algorithm improves the overall performance of the Hadoop system and that this algorithm has similar data locality to the JASL algorithm.

Individual Performance of Each Workloads

Illustrates the individual performance of each jobs, and each job setup comprised nearly 10 GB of data. The average execution time of the DJASL was compared with that of the default Hadoop algorithm in Environment 1 the results revealed that the sorting type jobs registered a higher execution time than the other jobs did, and that the join type jobs exhibited a shorter execution time. However, as shown in when multiple data were batch processed, the execution time did not increase multiples in continuation of the experiment. For example, if we have double data size of workloads, but the execution time will increase less than two times. We allocated nearly 100 GB of data storage space for each request involving different jobs and processed them in batches. The following sections present the experimental results.

Performance and Data Locality of the JAS and JASL Algorithms

In some of the ten requests, the performance of the JAS algorithm was not superior to those of Hadoop and DMR because the JAS algorithm sets slots inappropriately. Therefore, ``the resource Utilizations of some *Task Tracker*s became overloaded, and some tasks could not be executed until resources were released. Hence, the execution times of these tasks increased, causing the performance of the JAS algorithm to decrease compared with those of Hadoop and DMR. However, to simulate real situations, the average execution times of all jobs over ten requests were derived. In the heterogeneous computing environment, average execution times of the JAS and JASL algorithms were shorter than those of Hadoop and DMR. Depicts the average execution times of Hadoop, DMR, and the JAS and JASL algorithms. Because a substantial difference was observed in the CPU and memory resources between the nodes in those Environments (Tables No. 1-3), the performance of the algorithms in Environment 2 was superior to that of the algorithms in the other environments. However, the difference in performance between the environments was small. The execution time of the JASL algorithm was longer than that of the JAS algorithm, but the data locality of the JASL algorithm was substantially greater than that of the JAS algorithm (Figure No.6).

Thus, the large amount of extraneous network transformation produced by the JAS algorithm can be reduced. Because of the large processing capability difference between the nodes in Environment 2, higher numbers of efficient nodes were assigned for higher numbers of tasks, reducing data locality. Illustrates the percentage execution time relative to Hadoop. In the four environments, the performance of the JAS algorithm improved by nearly 15%-18% compared with Hadoop and nearly 18%-20% compared with DMR. Moreover, the data locality of the JASL algorithm improved by nearly 25%-30% compared with the JAS in these environments.

Performance and Data Locality of the DJASL Algorithm

The *Job Tracker* occasionally inaccurately sets the slots when the JASL algorithm is applied, potentially reducing the performance. Hence, the DJASL algorithm includes two parameters, namely *CPU count* and *IO count*, which are used to ensure accurate slot settings. The *Job Tracker* resets slots according to threshold values, and differences in the threshold values cause performance results to vary. If a threshold value is too high (i.e., slots are set incorrectly when the DJASL is applied), the *Job Tracker* must wait for a long period to reset the slots. By contrast, if a threshold value is too low, the *Job Tracker* must reset slots frequently. Inappropriate threshold settings hinder the maximization of resource utilization and negatively affect te performance of the Hadoop system. Therefore, an experiment was conducted in this study to determine the values of various threshold settings.

The threshold was set to 100, 200, 300, 400, and 500. According to these five values, five requests were sent to Hadoop, and each request contained ten disordered jobs (five Word count and five Tera sort). According to Figure No.7, setting the threshold value to 300 yielded the optimal performance.

Because the DJASL algorithm can reset slots through a count mechanism, its performance was superior to that of Hadoop. On average, the performance of the DJASL algorithm was superior to that of DMR. However, the performance of the DJASL algorithm was occasionally inferior to that of DMR because slots must be reset. In some scenarios, tasks executed by *Task Tracker*s are not removed by the *Job Tracker*. Therefore, the *Job Tracker* must wait for such tasks to be completed. When *Task Tracker*s become overloaded, the contained tasks cannot be completed until resources are released. Therefore, the execution times for these tasks are prolonged, reducing the performance of the DJASL algorithm compared with that of DMR. When the slots of the *Job Tracker* have been reset, the resources of each *Task Tracker* can be used to improve the performance of the Hadoop system. The average execution time of all jobs was used to simulate real situations.

We implemented three heterogeneous computing environments (Tables No.1-3) and compared each of them in detail with all the presented algorithms (e.g., Hadoop, DMR, JAS, JASL, DJASL). Figure No.8 (a) shows a comparison of the performance of the DJASL algorithm in Environment 2, which comprised a higher number of CPUs in slave computers compared with the master computers, and Environment 1. Figure No.8 (b) depicts a comparison of the performance of the DJASL algorithm in Environment 3 in which more memory was allocated to the slave computers compared with the master computers, and Environment 1. Figure No.8 (c) depicts a comparison of the performance of the DJASL algorithm in Environment 4, in which a higher number of CPUs and memory was allocated to the master node compared with the slave node, and Environment 1. A comparison of the results in Figure No.8 revealed that the numbers of CPUs demonstrated a considerably greater effect on performance regarding the amount of memory resources and improved processing capability of the master node. As shown in Figure No.8, the performance of the DJASL algorithm improved by approximately 27%-32% compared with DMR and by approximately 16%-21% compared with Hadoop.

The four heterogeneous computing environments were compared, and illustrates the results. The data locality of the DJASL algorithm was nearly identical to that of the JASL algorithm. In these environments, the JASL and DJASL effectively improved the data locality and also reduced the differences between these algorithms and Hadoop.

Table No.1: Heterogeneous CPU experimental environment

S.No		Master		Slave	
		Quantity	Specification	Quantity	Specification
1	Environment1	1	2cpu and 4GB memory	33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory
2	Environment 2	1	2cpu and 4GB memory	33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory

Table No 2: Heterogeneous RAM experimental environment

S.No		Master		Slave	
		Quantity	Specification	Quantity	Specification
1	Environment1	1	2cpu and 4GB memory	33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory
2	Environment 3	1	2cpu and 4GB memory	33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory

Table No 3: Heterogeneous Master experimental environment

S.No		Master		Slave	
		Quantity	Specification	Quantity	Specification
1	Environment1	1	2cpu and 4GB memory	33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory
2	Environment 4	1	4cpu and 8GB memory	33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory
				33	1 cpu and 2GB memory

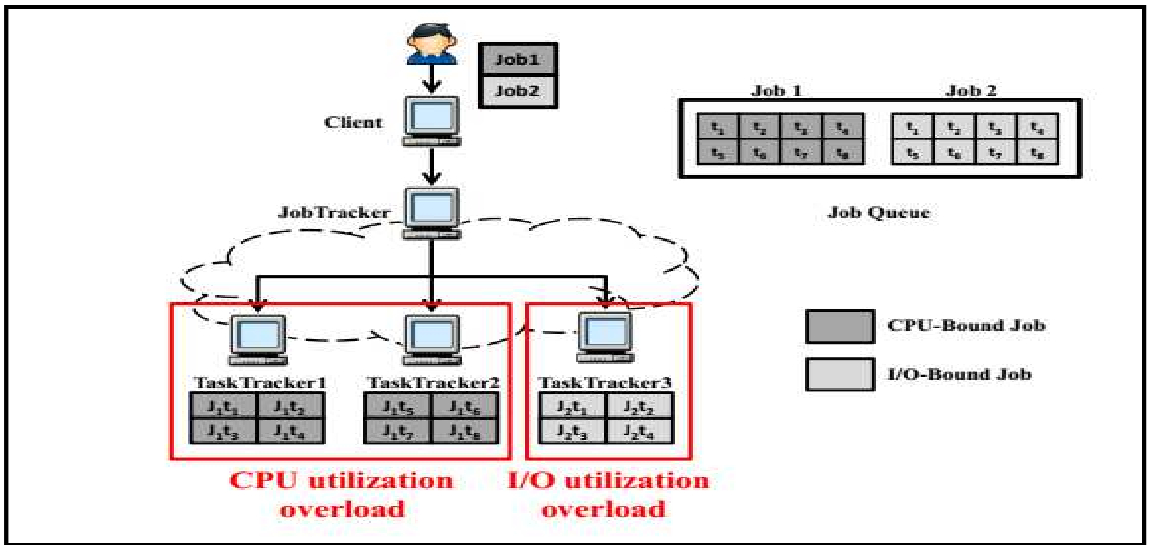
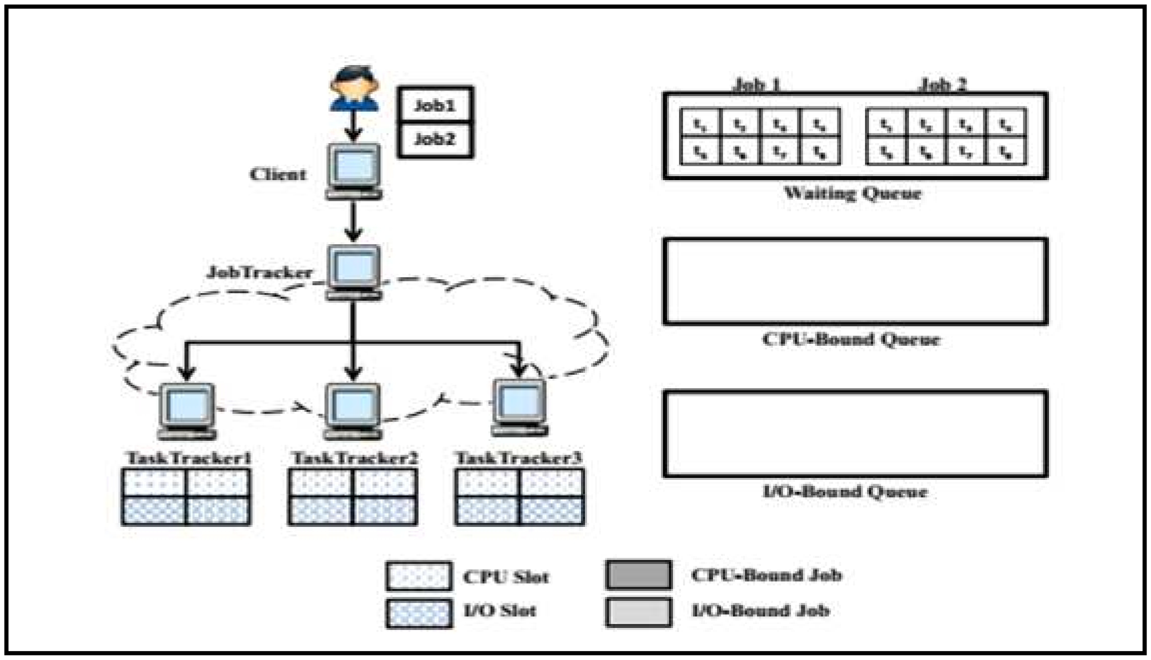
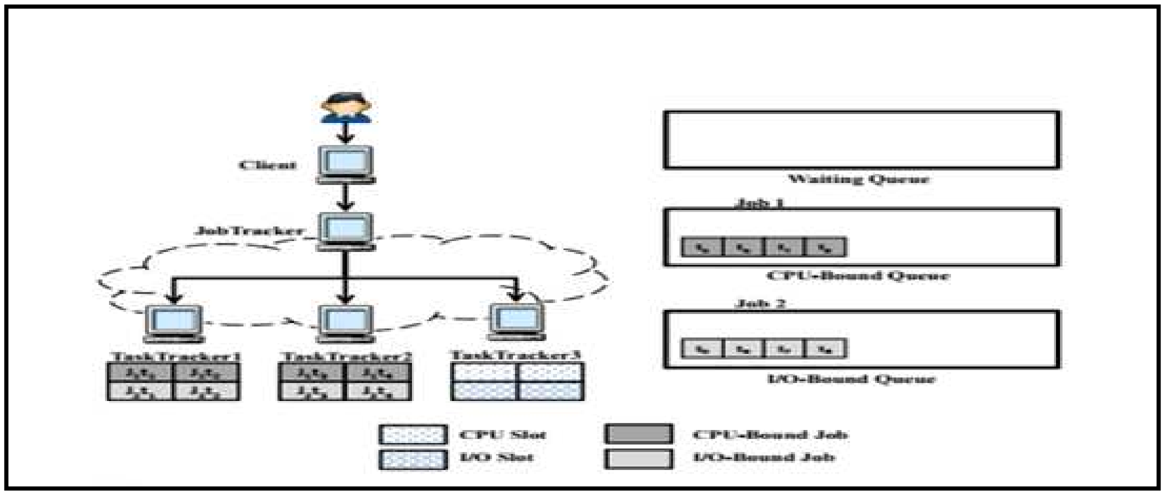


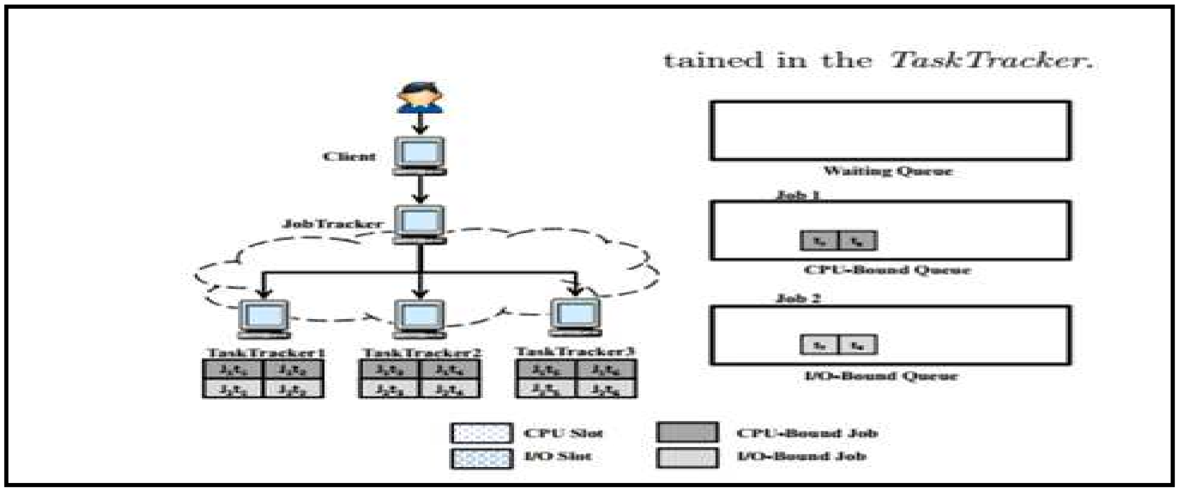
Figure No.1: Imbalanced resource allocation



(a) When a client submits a new job, the submitted job is added to the waiting queue. Subsequently, the scheduler classifies the job type



(b) After the scheduler classifies the job type, the jobs are added to the CPU-bound queue or the I/O-bound queue. The *Job Tracker* then assigns these tasks ac-cording to the number of free CPU or I/O slots con-tained in the *Job Tracker*



(c) The *Job Tracker* assigns tasks until all of the Task- Trackers have no free slots

Figure No.2: Workflow of a DMR scheduler

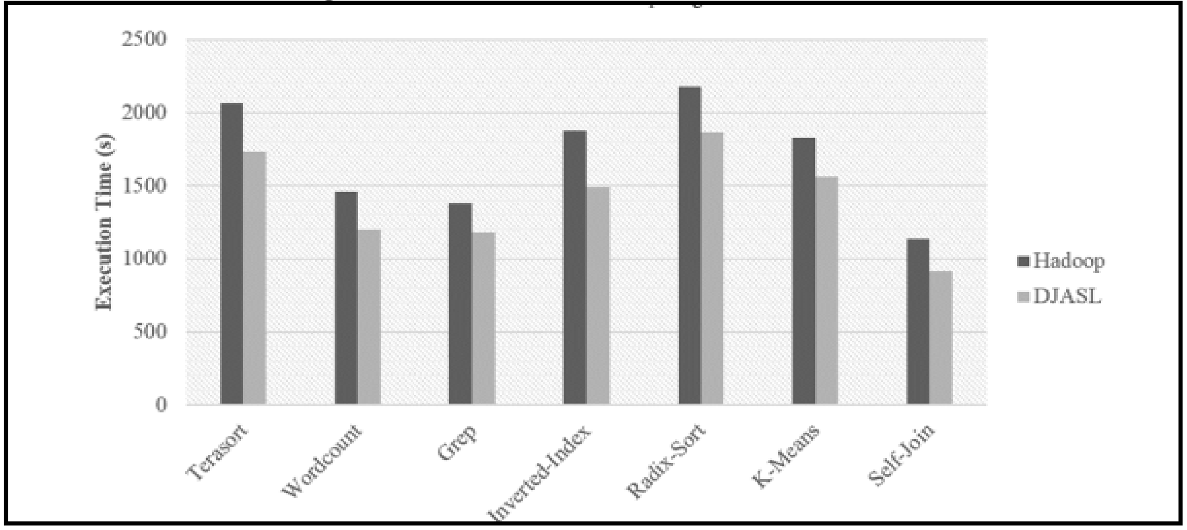


Figure No.3: Average execution time of each job in the Hadoop system and DJASL algorithm

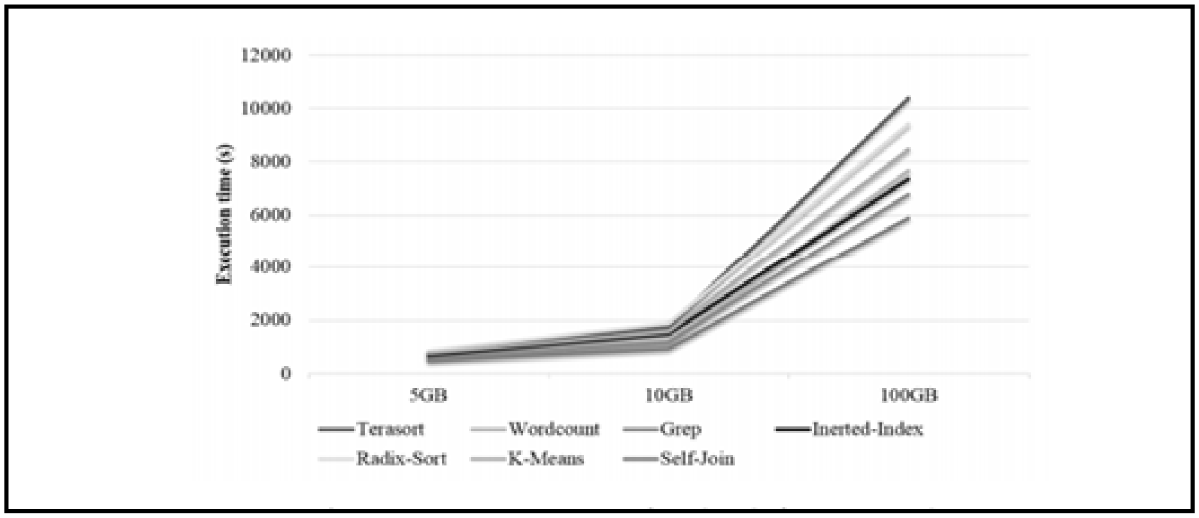


Figure No.4: Average execution time of each job for various data sizes

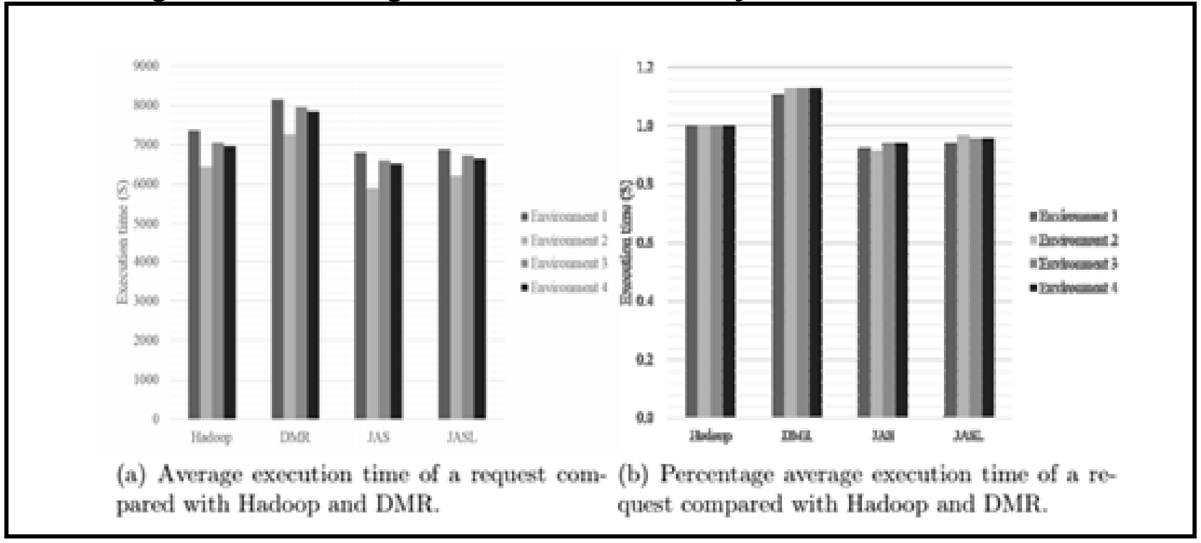


Figure No.5: Performance of JAS and JASL compared with Hadoop and DMR in four computing environments

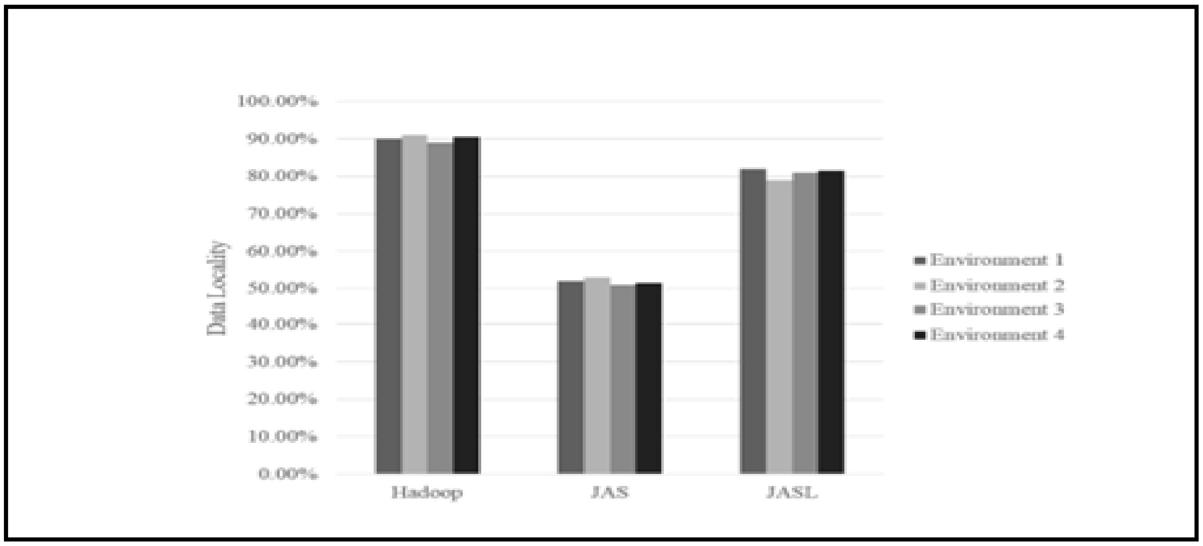


Figure No.6: Data locality of JAS and JASL compared with Hadoop in four environments

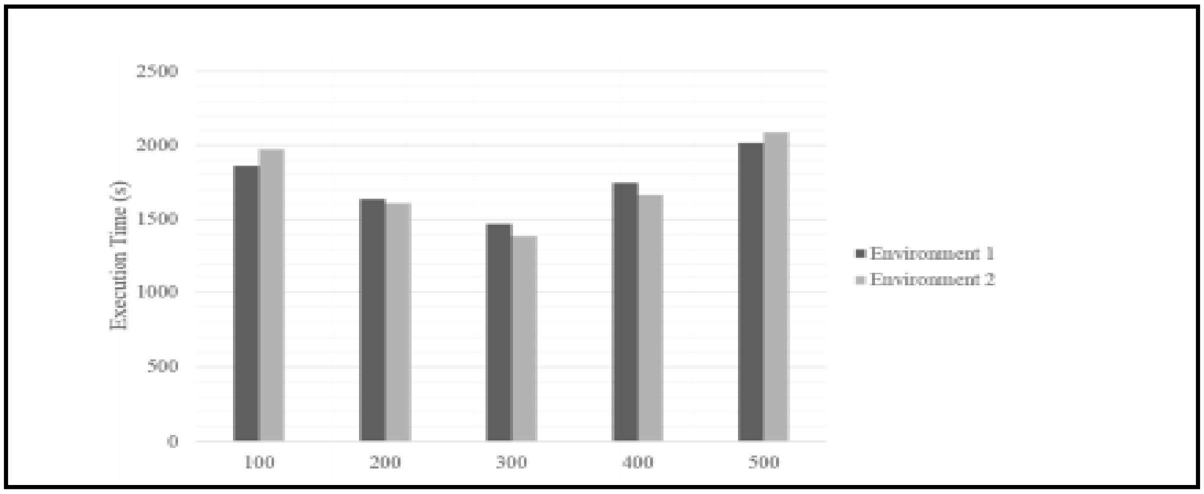


Figure No.7: Average execution time of a job in DJASL for setting the various thresholds

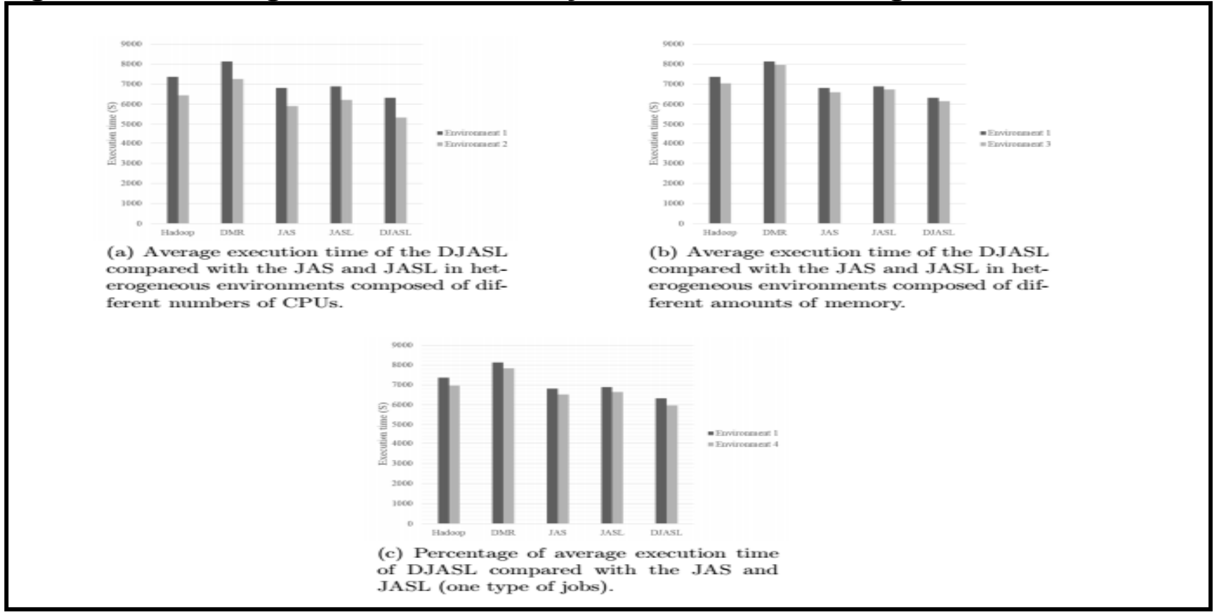


Figure No.8: Performance of the DJASL compared with the JAS and JASL in different heterogeneous computing environments

CONCLUSION

This paper proposes job scheduling algorithms to provide highly efficient job schedulers for the Hadoop system. Job types are not evaluated in the default job scheduling policy of Hadoop, causing some *Task Tracker*s to become overloaded. According to the proposed DJASL algorithm, the *Job Tracker* first computes the capability of each *Task Tracker* and then sets the numbers of CPU and I/O slots accordingly. In addition, the DJASL algorithm substantially improves the data locality of the JAS algorithm and resource utilization of each *Task Tracker*, improving the performance of the Hadoop system. The experimental results revealed that performance of the DJASL algorithm improved by approximately 18% compared with Hadoop and by approximately 28% compared with DMR. The DJASL also improved the data locality of the JAS by approximately 27%. The proposed scheduling algorithms for heterogeneous cloud computing environments are independent of systems supporting the Map Reduce programming model. Therefore, they are not only useful for Hadoop as demonstrated in this paper, but also applicable to other cloud software systems such as YARN and Aneka.

ACKNOWLEDGMENT

This research was partly supported financially by the Headquarters of University Advancement at National Cheng Kung University, which is sponsored by the Ministry of Education, Taiwan, R.O.C.

CONFLICT OF INTEREST

We declare that we have no conflict of interest.

BIBLIOGRAPHY

1. Apache Hadoop. http://hadoop.apache.org/
2. Apache Hadoop YARN. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.
3. Hadoop's Capacity Scheduler. http://hadoop.apache.org/core/docs/current/capacity scheduler.html.
4. Matei Zaharia. "The Hadoop Fair Scheduler"http://developer.yahoo.net/blogs/hadoop/FairSharePres.ppt.
5. Ahmad F, Chakradhar S T, Raghunathan A and Vijaykumar T N. "Tarazu: optimizing mapreduce on heterogeneous clusters," *In ACM SIGARCH Computer Architecture News*, 40(1), 2012, 61-74.
6. Atallah M J, Lock C, Marinescu D C, Siegel H J and Casavant T L. "Co-scheduling compute-intensive tasks on a network of workstations: model and algorithms," *In Proceedings of the 11th International Conference on Distributed Computing Systems*, 11th, 1991, 344-352.

Please cite this article in press as: Indumathi J. Novel scheduling algorithms for efficient deployment of map reduce applications in heterogeneous computing environments, *International Journal of Engineering and Robot Technology*, 3(2), 2016, 48 - 57.