BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **17**, No 1
Sofia • 2017 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2017-0012

Empirical Study of Job Scheduling Algorithms in Hadoop MapReduce

Jyoti V. Gautam[1], *Harshadkumar B. Prajapati*[1], *Vipul K. Dabhi*[1], *Sanjay Chaudhary*[2]

[1]*Department of Information Technology, Dharmsinh Desai University, Nadiad, India*
[2]*Institute of Engineering & Technology, Ahmedabad University, Ahmedabad, India*
Emails: jyotiii.gautam16290@gmail.com prajapatihb.it@ddu.ac.in vipul.k.dabhi@gmail.com sanjay.chaudhary@ahduni.edu.in

Abstract: *Several Job scheduling algorithms have been developed for Hadoop-MapReduce model, which vary widely in design and behavior for handling different issues such as locality of data, user share fairness, and resource awareness. This article focuses on empirically evaluating the performance of three schedulers: First In First Out* (*FIFO*)*, Fair scheduler, and Capacity scheduler. To carry out the experimental evaluation, we implement our own Hadoop cluster testbed, consisting of four machines, in which one of the machines works as the master node and all four machines work as slave nodes. The experiments include variation in data sizes, use of two different data processing applications, and variation in the number of nodes used in processing. The article analyzes the performance of the job scheduling algorithms based on various relevant performance measures. The results of the experiments are evident of the performance being affected by the job scheduling parameters, the type of applications, the number of nodes in the cluster, and size of the input data.*

Keywords: *Big Data, Hadoop, MapReduce, job scheduling, analysis, experimental evaluation.*

1. Introduction

For today's era of the digital world, generating data rapidly, MapReduce [1] provides an ideal framework for the processing of such large data by using parallel and distributed programming approaches. In the MapReduce, a computation is divided into two functions: Map and Reduce. These two functions can be modified based on the type of processing needed by an application. The MapReduce works with Hadoop [2] for processing a large number of datasets, which may be structured or unstructured. The architecture needed for processing using MapReduce consists of one Master node (running JobTracker) and many Slave nodes (running TaskTrackers). Hadoop is an open-source framework that enables distributed processing of large datasets through its distributed file system and distributed

processing framework. Hadoop also provides a reliable shared storage system called as Hadoop Distribute File System (HDFS). HDFS provides access to files in the cluster [3]. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS is implemented by two components: NameNode and DataNodes. The NameNode component, present at a master node, manages the file system namespace and is a centralized service. The DataNodes component stores the blocks of data. Another component Secondary NameNode is responsible for periodic checkpointing.

As Hadoop is a new framework, it still requires improvements in few aspects [4]. One of improvement directions is Job scheduling. However, Job scheduling in Hadoop is complex and it gets affected by many parameters; therefore, the study of existing job scheduling is needed before any attempt is made for any improvement. Job scheduling controls the order of tasks to be run and the allocation of resources, which directly affects the overall performance of the applications and system resource utilization of Hadoop cluster. The performance of job scheduling in the cluster depends on upon the job configurable parameters, scheduler configurable parameters, cluster configuration, and input data. There are more than 180 parameters in the Hadoop. A few important parameters include the number of replicas of input data and the number of parallel map/reduce tasks to run. Configuring various job scheduling parameters in order to get overall good performance from the system is still a challenging task.

Existing research works, e.g., [5, 6], show that the performance of MapReduce applications depends on the cluster configuration, job configuration settings, and input data. In their work, First In First Out (FIFO) scheduler was used for experiments. The work in [7] presents an evaluation among FIFO, Fair, and Fair with delay scheduling. Adding delay time to Fair scheduler increase the processing of jobs. In [8], Cloud benchmark suite CloudRank-D is used to evaluate different Hadoop job schedulers, which include FIFO scheduler, Fair scheduler, Fair scheduler with delay, Capacity scheduler, and Hadoop On Demand (HOD).

This article focuses on the experimental evaluation of schedulers, which are present in Hadoop, namely FIFO, Fair, and Capacity scheduler. To perform experiments, we have created our own Hadoop cluster of 4 nodes, which has one master and four slaves. The performance of these three schedulers is compared by using performance metrics, which are CPU/processing time, Turnaround time, Data processed per second, and Data locality rate. Our goal is to demonstrate empirically, based on a real testbed not on a simulation, that different parameters, workloads, and different applications play a significant role in the performance of the schedulers as well as on the system.

The structure of this article is as follows. Section 2 presents background and discusses related work. Section 3 presents preparation of Hadoop cluster testbed, which we use for carrying out experiments. Section 4 discusses experiment settings including Hadoop cluster setup, performance metrics, and input dataset. Section 5 presents empirical evaluation focusing on the results and their analysis. Finally, Section 6 presents the conclusion and provides future directions.

2. Background and related work

Hadoop is a widely used open-source implementation of Google MapReduce [1]. Generally, many commodity computers are used to build a MapReduce cluster, in which one computer acts as the master node and others as slave nodes. A Hadoop cluster uses HDFS to manage its data. HDFS divides each file into a small fixed-size (e.g., 64 MB) blocks and stores several (e.g., 3) replicas of each block in local disks of cluster machines. In a MapReduce job, input files are read from HDFS and a Map task is preferably scheduled on a machine that contains a replica of the corresponding input data. MapReduce uses the replication feature of HDFS for improving the performance. MapReduce uses terms job and task. A MapReduce job is composed of many tasks, in which a task carries out either map or reduce processing. The job scheduler runs on the JobTracker node; it plays an important role in deciding where the tasks of a particular job will be executed in the cluster. Three widely used Job schedulers in Hadoop are as follows.

2.1. Schedulers

This sub-section briefly discusses salient characteristics of three schedulers: FIFO, Fair, and Capacity schedulers.

2.1.1. FIFO scheduler

The default scheduler in Hadoop is FIFO [7]. In FIFO scheduler, the jobs are submitted to a single queue and are executed sequentially. For assignment of tasks, the scheduler follows the strict FIFO job order. FIFO does not allocate any task from other jobs if the first job in the queue still has an unassigned map task. The drawback of this scheduler is that the strict FIFO job order reduces data locality, and only after completion of the previous job, next jobs in the job queue are assigned to nodes.

2.1.2. Fair scheduler

The Fair scheduler [9, 10] is developed by Facebook. The fair scheduler assigns resources to each job such that on average each job gets an equal share of available resources. The Fair scheduler groups jobs into named pools based on the configurable attributes such as user name and UNIX group, and then the scheduler performs fair sharing between these pools. Each pool has a guaranteed capacity i.e., the minimum number of Map and Reduce slots. By default, all pools have equal shares of resources (Map and Reduce task slots). The fair scheduler also supports pre-emption of tasks. The jobs that require less time for execution can access CPU early. If there is only one job running in the cluster, it gets resources of the entire cluster. If any pool is not using its slots, then these idle slots can be used by other pools.

2.1.3. Capacity scheduler

The Capacity scheduler [9, 11] is developed by Yahoo. The Capacity scheduler is designed for multiple organizations sharing a large cluster. The Capacity scheduler provides a minimum capacity and shares excess capacity among users. A capacity is assigned to each queue. Several queues are created each with configurable Map and

Reduce slots. The Capacity refers to the size of Map and Reduce slots. Within each queue, jobs are scheduled in FIFO order. When a TaskTracker slot becomes free, the queue with minimum load is chosen, from which the oldest remaining job is chosen. The Capacity scheduler supports the priority of jobs. Jobs with higher priority have access sooner to resources as compared to jobs with lower priority. The priority of the job is adjusted based on the time the job was submitted. The overall capacity of the cluster is the sum of capacities of each queue. If any of the queue's allocated capacity is unused, then this capacity is assigned to other queues. The Capacity scheduler has the ability to control the allocation of resources based on physical machine resources. The Capacity scheduler understands the scheduling tasks based on memory consumption of job's task.

2.2. Related work

Job scheduling algorithms play an important role in the performance of the Hadoop-based system. In recent past, various studies with experimental and theoretical aspects of job scheduling and other related topics have been reported in the literature.

G u et al. [12] carried out a detailed analysis of MapReduce execution environment and showed vital issues of the execution environment. Their analysis suggested and empirically demonstrated the use of instant messaging mechanism instead of existing heartbeat messages. The work in [13] proposed enhancement to Hadoop MapReduce framework; existing MapReduce framework can run well in homogeneous clusters, not in heterogeneous clusters. Their work proposed the enhancements in the design of MapReduce framework that can handle node heterogeneity during phases such as data distribution, task scheduling, and job control. A theoretical and experimental examination on scheduling in MapReduce was reported in [14]. Their work formulated MapReduce server-job organizer problem and showed that it is NP-complete. As a solution to this problem, their work initially developed 3-approximation algorithm, and then proposed a fast heuristic algorithm. Their work carried out an evaluation of the proposed algorithm via simulation and real system, on Amazon EC2. L i, J i a n g and R u i z [15] modelled the problem of makespan minimization of periodical batch jobs as a two-stage hybrid flow-shop scheduling problem. Their work designed three heuristics to solve the problem and showed further improvement using data locality of tasks.

Energy saving without degradation in the performance of the system is an important research direction. A recent research in [16] proposed two energy-aware MapReduce scheduling algorithms that reduce energy cost incurred without violation of SLA in Hadoop clusters. Some usage scenarios may demand completing jobs before the specified deadline. In this direction, T a n g et al. [17] proposed MapReduce Task Scheduling algorithm for Deadline constraints in Hadoop platform (MTSD) that allows a user to specify the deadline for completing a job. The proposed algorithm meets the deadline by using the proposed data distribution model that distributes data based on nodes' capacity level, which it detects using node classification algorithm. In pay-as-you-go model, customers may want to finish their jobs within specified budgets, and possibly before deadlines. A study in [18]

modelled a batch of MapReduce jobs as a workflow and scheduled the jobs on heterogeneous VMs in a cloud with satisfying budget and deadline constraints.

In a computer network based distributed system such as Hadoop, the probability of failure of any entity cannot be ignored. Such assumption is considered in the research work [19]. Their work proposed replication-based fault tolerance mechanism that addresses node and task failure and showed that their replication-based fault tolerance mechanism outperforms existing rescheduling-based fault tolerance of Hadoop system. Another related research work considers the decision of starting backup tasks on appropriate nodes. Chen et al. [20] proposed History-based Auto-Tuning (HAT) MapReduce scheduler that accurately estimates the progress of tasks to allow launching of backup tasks on appropriate nodes. Scalable computing on Cloud was attempted by Gunarathne et al. [21]. Their work proposed Twiser4Azure, which is an iterative MapReduce runtime for Windows Azure Cloud. Twiser4Azure allows novel programming model to solve problems such as Multi-Dimensional Scaling and K-Means Clustering that can involve iterative MapReduce computations. Their work also included a decentralized cache aware task scheduling to provide fault tolerant execution of computations.

As a Hadoop system allows distributed storage of data, efficient access to data can improve the performance of applications. A research work in [22] proposed HPSO (High-Performance Scheduling Optimizer) that enhances the scheduling algorithm based on pre-fetching of data needed to map tasks. This pre-fetching reduces the time needed to transfer data over the network and hence improves the performance. Some scientific data processing applications, e.g., data analytics, need to process a very large amount of data. In this direction, the Sehrish et al. [23] proposed a MapReduce-based framework to support execution of HPC analytics applications, which generally need transferring of large data from an HPC storage system to a data intensive computing system, i.e., MapReduce. Their proposed framework eliminates multiple scans of data. Their work also implemented a data-centric scheduler for improving the performance of HPC analytics applications. Another related work for the scientific domain was reported in [24]. Their work proposed G-Hadoop, which is an extension of MapReduce framework, that allows the use of nodes of distributed data centres to solve very large-scale data analysis scientific applications such as used in High Energy Physics (HEP) and Large Hadron Collider (LHC) experiments.

Survey works help researchers in understanding the relevant concepts and the state-of-the-art. A study in [25] proposed a novel multidimensional classification framework to classify scheduling algorithms used in MapReduce. The classification is in three dimensions: (1) quality requirements of a MapReduce system; (2) scheduling entities; (3) adaptation to dynamic environments. Their work analysed various scheduling algorithms used in MapReduce in depth based on the proposed classification framework. Another composition in [26] surveyed various works focused on distributed data management and data processing using MapReduce framework. Their study provided comprehensive coverage of MapReduce system, various implementations of MapReduce, MapReduce implementation of database operators (e.g., join operations), and database systems using MapReduce processing.

Their study also discussed possible optimizations, i.e., extensions to MapReduce, for scheduling. A survey work in [27] presented a survey of various technical aspects of MapReduce framework with a focus on advantages and disadvantages. Their work also discussed various schedulers used for MapReduce jobs. Recently, G a u t a m et al. [9] presented a survey and analysis of various schedulers used in Hadoop. Furthermore, their work also carried out a survey and analysis of various customized MapReduce frameworks. A research article in [28] carried out a survey of systems in HPC, cloud, and Big Data environments with a focus on performance and energy efficiency. Their work surveyed characterization of workload for such systems. Their extensive review of various research works suggested that more than 50% papers focus on improvement of performance and energy efficiency.

An empirical evaluation of any system can provide a performance comparison of a system with other related works. Furthermore, critical analysis of a system can suggest possible improvements in various entities of the system. In this direction, A l t h e b y a n et al. [29] empirically evaluated MapReduce scheduling algorithms such as FIFO, Matchmaking, Delay, and MultiThreading Locality (MTL) using CloudExp simulator, which is an extension of CloudSim. Our work differs from their work in the way that we carry out an experimental evaluation on a real Hadoop cluster, rather than a simulation.

3. Development of a testbed for evaluation of job scheduling algorithms

We implement our own Hadoop cluster testbed consisting of four machines. This section discusses deployment of the testbed, presents challenges faced while deploying the testbed, and discusses the steps for configuring schedulers.

3.1 Deployment of the testbed

Hadoop is a framework written in Java for running applications on large clusters of commodity hardware, and it incorporates features similar to Google File System (GFS). Hadoop is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets. Hadoop has master/slave architecture in which one machine is configured as master and other machines are configured as slaves. We have implemented our own testbed having four nodes. The architectural diagram of our testbed is shown in Fig. 1. The diagram also shows hardware and software configuration of used machines.

Hadoop framework can be installed on windows OS, Mac OS, and on Linux OS. We have installed Hadoop on Cent OS. As a prerequisite, Hadoop installation requires Java on all the machines of the cluster. We also configure password-less SSH access to all the machines of the testbed, which is a requirement of Hadoop cluster. Hadoop master uses the password-less login for initiating process execution remotely. In addition to these requirements, there are a few files in Hadoop which we need to configure. We configure following files: core-site.xml, mapred-site.xml, hdfs-site.xml, masters, and slaves. All these files are present in conf. directory under

Hadoop base directory. All machines must be able to reach each other over the network; therefore, we put all four machines on the same network, for which we use a network switch.

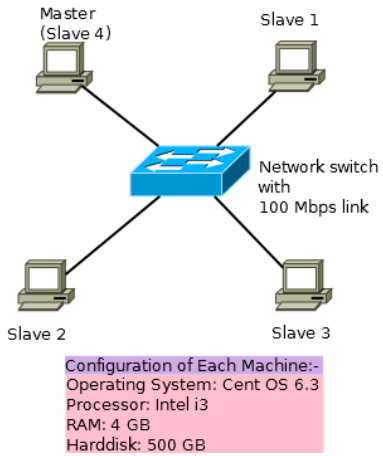

Fig. 1. Architecture of our Hadoop cluster testbed

3.2. Challenges we faced while installing Hadoop

We faced a few problems while installing Hadoop. We briefly discuss these problems and their solutions.

3.2.1. Problem with ssh configuration

Error: connection refused to port 22:
This is the most common error while installing Hadoop. To resolve this error, make sure that the machine on which you are trying to login should have ssh server installed. Make sure that the keys are configured properly and the public key is shared with the machine that you want to login to. If the problem persists then check for configuration of ssh on the machine; the configuration is kept in /etc/ssh/sshd_config file.

3.2.2. NameNode is not reachable

Error: Retrying to connect 192.x.x.x:
Check whether daemon process is running or not. Check for core-site.xml file in conf directory of Hadoop. Check for value for property hadoop.tmp.dir. It should be set to a path where the user that is trying to run Hadoop has write permissions. We need to format the NameNode again and need to start Hadoop again.

3.2.3. No DataNode to stop

Error: java.io.IOException: Incompatible namespaceIDs.
No datanode to stop error occurs while stopping the DataNode. In this situation, we need to stop the problematic DataNode(s). Then, we need to edit the value of NamespaceID in {dfs.data.dir}/current/VERSION to match the corresponding value

of the current NameNode in {dfs.name.dir}/current/VERSION. Finally, we need to restart the fixed DataNode(s).

3.2.4. Connection error between DataNode and NameNode

Error: DataNode cannot connect to NameNode:
This error occurs when DataNode is not able to connect with NameNode. To solve this error, we need to stop the iptables, which is a firewall in Linux. Iptables is a rule based firewall system and is normally pre-installed on a Unix/Linux operating system. Iptables controls the incoming and outgoing packets.

3.2.5. Login error while logging to Slave machine from Master machine

Error: This is an SSH login error:
This error occurs while logging into machines that are remotely accessible. For solving this problem, after generating and distributing keys for passwordless login, we need to place the public key on all slave machines in the home directory of the Hadoop cluster user.

3.3. Scheduler configuration

For running different schedulers in Hadoop, a few properties have to be configured in the configuration files. The schedulers that can be configured are Fair and Capacity schedulers. FIFO scheduler does not need to be configured.

3.3.1. Steps for configuring Fair scheduler

The Fair scheduler is available as a JAR file in the Hadoop under the contrib/fairscheduler directory. The name of the JAR file, version specific, would be of the form hadoop-*-fairscheduler.jar.

To run the Fair scheduler in Hadoop cluster, we need to put its jar file on the CLASSPATH. The other way is to copy the hadoop-*-fairscheduler.jar file from Hadoop directory to HADOOP_HOME/lib.

To allow Hadoop framework use the Fair scheduler, we need to set up the following property with indicated value in the site configuration:

- Property: mapred.jobtracker.taskScheduler
- Value: org.apache.hadoop.mapred.FairScheduler

We need to define fair scheduler allocations file (location specified in maped-site.xml) to use the queues and we need to assign resources to these queues.

Two pools were created for processing of jobs. Pre-emption is enabled. For finishing short jobs faster, weight booster was enabled. The interval at which to update fair share calculations is kept 500 ms.

3.3.2. Steps for configuring Capacity scheduler

The Capacity scheduler is available as a JAR file in the Hadoop under the contrib/capacity-scheduler directory. The name of the JAR file, version specific, would be of the form hadoop-capacity-scheduler-*.jar.

To run the Capacity scheduler in Hadoop cluster, we need to put the JAR file on the CLASSPATH. The other way is to copy the hadoop-capacity-scheduler-*.jar from Hadoop directory to HADOOP_HOME/lib.

To allow the Hadoop framework use the Capacity scheduler, we need to set up the following property with the indicate value in the site configuration:

- Property: mapred.jobtracker.taskScheduler
- Value: org.apache.hadoop.mapred.FairScheduler

We define multiple queues to which users can submit jobs using the Capacity scheduler. To define multiple queues, we use the mapred.queue.names property in conf/mapted-site.xml.

Several properties of the queue can be configured to control the behaviour of scheduler. This configuration is in the conf/capacity-scheduler.xml.

Two queues are created for processing of the job. Queues with 50-50% capacity and also with 10-90% capacity.

4. Experiment setup

This section discusses the experimental setup used for evaluating FIFO, Fair, and Capacity schedulers. This section also discusses workload and performance measures used for evaluation.

4.1 Experimental environment

To allow the readers be able to replicate the experiments, we briefly discuss our experiment environment. As discussed earlier, the testing environment consists of four nodes (one master and four slaves). The master node was designated to run JobTracker and NameNode, while the slaves run TaskTrackers and DataNodes (slave daemon of HDFS). The master node is also configured to exhibit the functionality of a slave node. Each node has Intel core i3, 3.4 GHz processors, with a single hard disk capacity of 500 GB, 4 GB RAM, and 100 Mbps network connection. All the nodes were installed with CentOS 6.3 operating system with Java 1.6.0. 24 (Open JDK), executing Hadoop 1.2.1 (stable release). Table 1 shows the Hadoop configurable parameters.

Table 1. Hadoop parameters

Parameters	Details (configured value)
HDFS block size	64 MB
Speculative execution	Enabled
Heartbeat interval	3 s
Number of map tasks per node	2
Number of reduce tasks per node	1
Replication factor	2
HDFS block size	64 MB
Speculative execution	Enabled
Heartbeat interval	3 s

4.2. Workload and performance measures for experiments

We choose two representative Hadoop applications: WordCount and Grep [30]. WordCount has been used by many researchers as a benchmark MapReduce application for such experiments. WordCount is a CPU-intensive application, which reads text files and counts how often the words occur. Another application – Grep is frequently used in data mining algorithm; it extracts matching strings from text files. For these two MapReduce applications, we use input data files of sizes 818.5 MB, 1.2 GB and 2.4 GB, which are text files that we manually generated by copying text content from various technical documents. Text files of varying sizes are used to observe the effect of data size on the performance of the schedulers.

For measuring the performance of scheduler, following performance measures are used.

CPU Time: It indicates the amount of time for which a CPU is used for processing instructions of a program.

Data processed per second [30]: It is defined as the input data size divided by the application running time.

Turnaround Time [8]: It is measured as the total time taken from the submission of a job until the end of the job execution.

Data Locality [31]: It is defined as the number of tasks running on the same resources where their stored data are located.

The CPU Time can be obtained from the terminal output, after the completion of a job or from the JobTracker web interface provided by Hadoop framework. The Turnaround Time is obtained by executing the command "Hadoop job – history" in the terminal as well as from the JobTracker web interface. The Data Locality rate can be obtained from the JobTracker web interface as well as from a terminal.

5. Empirical evaluation: results and analysis

This section provides the results of our experiments. In each experiment, we compare FIFO, Fair, and Capacity schedulers. The comparison is based on the earlier discussed performance metrics.

5.1. Comparison based on CPU Time

Comparison of CPU Time used by a job is done by varying the size of data given to the three schedulers. As shown in Fig. 2, the default FIFO scheduler takes more CPU Time as compared to Fair and Capacity schedulers. Among the three schedulers, Fair scheduler takes less CPU Time. Fair scheduler supports pre-emption of tasks while Capacity scheduler does not support pre-emption. If a pool has not received its fair share for a certain period, then the Fair scheduler will kill tasks in pools running beyond the capacity in order to give the slots to the pool running under capacity. This pre-emption was enabled may be because of which Fair performs well as compared to other two schedulers. Furthermore, there is a minor difference between the processing time of Fair and Capacity schedulers. Both the schedulers have the property of assigning some capacity or slots to the Map and Reduce tasks; while FIFO schedule does not have this property. The capacity or slots are considered as resources

(Map and Reduce slots) in the cluster, which are assigned to the queues. Queues are configured with guaranteed Map and Reduce slots. If more slots are available, then the execution time will decrease. More capacity means more Map and Reduce slots, thus it decreases Map execution time for a job in a queue.

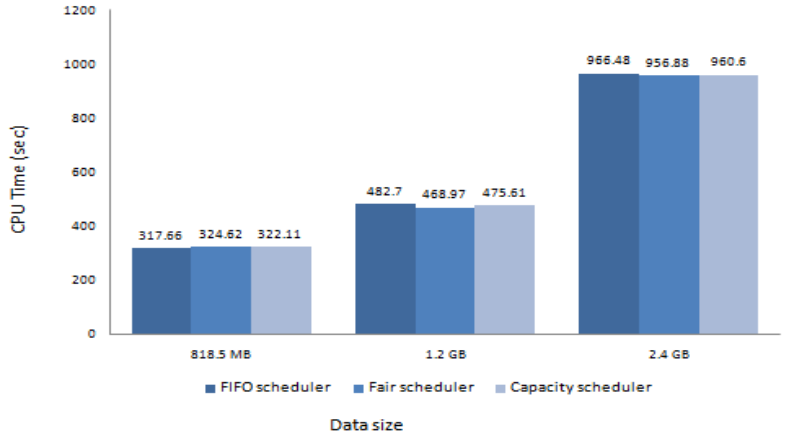

Fig. 2. Effect of data size on CPU Time (s) of Wordcount job for FIFO, Fair, and Capacity schedulers

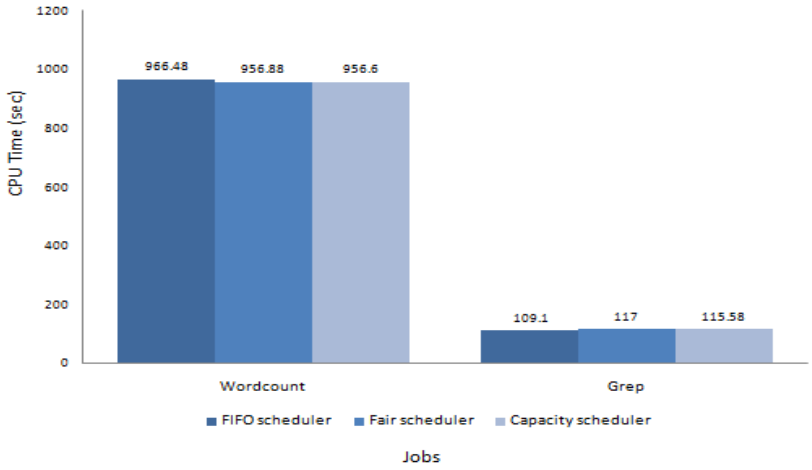

Fig. 3. Comparison of CPU Time of Wordcount and Grep jobs for FIFO, Fair, and Capacity schedulers for processing of 2.4 GB of data

The CPU Time also varies according to the type of application executed on the system. As shown in Fig. 3, Wordcount takes more CPU Time as compared to Grep. Wordcount takes more CPU Time because it is a CPU intensive job. There are minor differences in the processing times of all the three schedulers. As shown in Fig. 3, the processing time of Wordcount is 8.2 times higher than that of running Grep, when they both process 2.4 GB of dataset on FIFO scheduler, Fair scheduler, and Capacity scheduler. Therefore, running the same application with different schedulers may not produce large differences.

5.2. Comparison based on Turnaround Time

Comparison of Turnaround Time of a job is done by varying the size of data given to the three schedulers. The Turnaround Time often shows an intuitive view of system performance. Among three schedulers, Fair scheduler reduces the job Turnaround Time. It is mainly because this scheduler can reduce the network cost. When data size is large, e.g., 2.4 GB, Fair and Capacity schedulers take less Turnaround time as shown in Fig. 4.

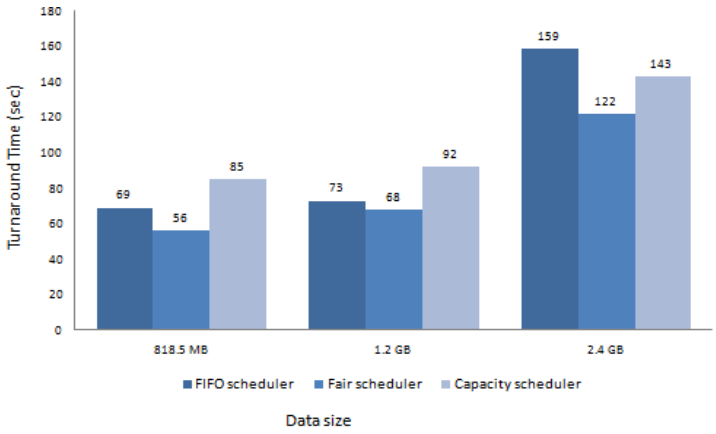

Fig. 4. Effect of data size on Turnaround Time (sec) of grep job for FIFO, Fair, and Capacity scheduler

Fair and Capacity schedulers can limit the number of concurrent jobs. This limitation can reduce the cost of scheduling between jobs in the task pool, and system performance gets improved. The Data locality is also an important property for reducing the Turnaround Time. If the data is nearer to the computation node, then waiting time of a job gets reduced and network cost will also get reduced.

5.3. Comparison based on Data processed per second

Comparison of Data processed per second is done by varying the size of data given to the three schedulers. As shown in Fig. 5, for input data sizes 1.2 GB and 2.4 GB, Fair and Capacity schedulers process more bytes as compared to FIFO scheduler. The Capacity scheduler and the Fair scheduler can enhance the system utilization, which directly affects the processing of data. The Capacity scheduler provides capacity guarantees to queues, i.e., the maximum number of Map and Reduce slots. The unused capacities can be used by other queues. Consequently, as the cluster utilization increases, the execution time of a job decreases, and data processing capability increases. In Fair scheduler, jobs are submitted to the pools. Pools can also be considered as queues. Each pool is provided with a minimum number of Map/Reduce slots. When there are pending jobs in the pool, it gets at least minimum number of slots assigned to it. However, if the pool has no jobs, the slots can be used by other pools. As the unused Map and Reduce slots can be used by other jobs, the execution time will get reduced and Data processing capability is increased.

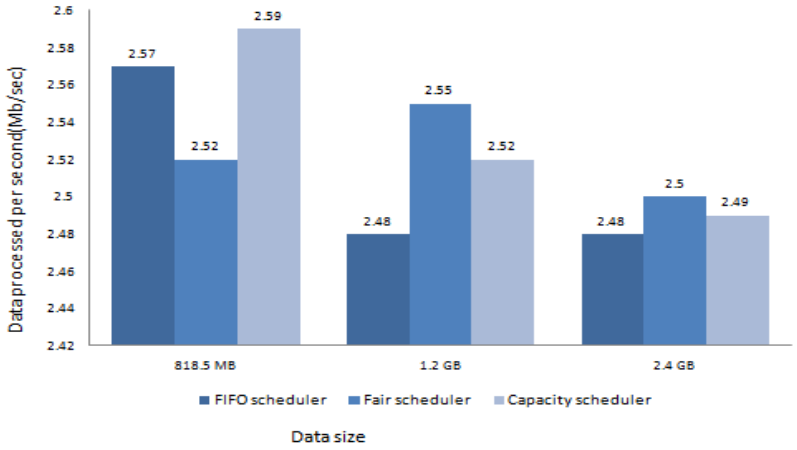

Fig. 5. Comparison of Data processed per second for Wordcount job for FIFO, Fair, and Capacity schedulers

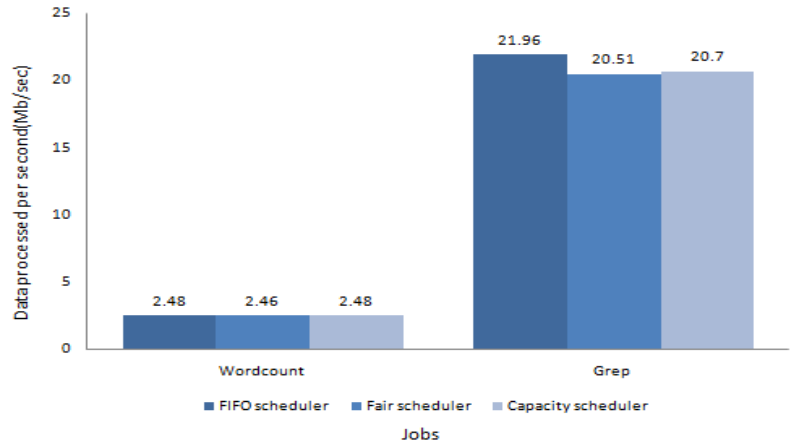

Fig. 6. Comparison of Data processed per second by Wordcount and Grep for FIFO, Fair, and Capacity schedulers for processing of 2.4 GB of data set

Processing of Data processed per second is also dependent on the type of application. Wordcount processes very fewer data per second as compared to Grep. From Fig. 6, we can conclude that the data processed per second of Grep is 8 times higher than that of running Wordcount when they both process 2.4 GB of data set on FIFO scheduler, Fair scheduler, and Capacity scheduler. This shows that schedulers' performance is also dependent upon the type of a job. Wordcount is CPU intensive job, so it takes more time in processing which directly affects data processing per second, whereas Grep takes less time for processing which directly increases the data processing per second.

5.4. Comparison based on the number of nodes

The number of nodes in a cluster plays a very important role in the performance of scheduler. Fig. 7 shows the effect of the number of nodes on CPU Time for processing

of 818.5 MB of data set by Wordcount job. From Fig. 7, it can be seen that as the number of nodes increases, the performance of Capacity scheduler increases. The capacity scheduler takes less processing time as the number of nodes increases. This is because Capacity scheduler utilizes cluster resources efficiently. It is basically designed for multiuser and multiple organizations sharing a cluster with guaranteed capacities.

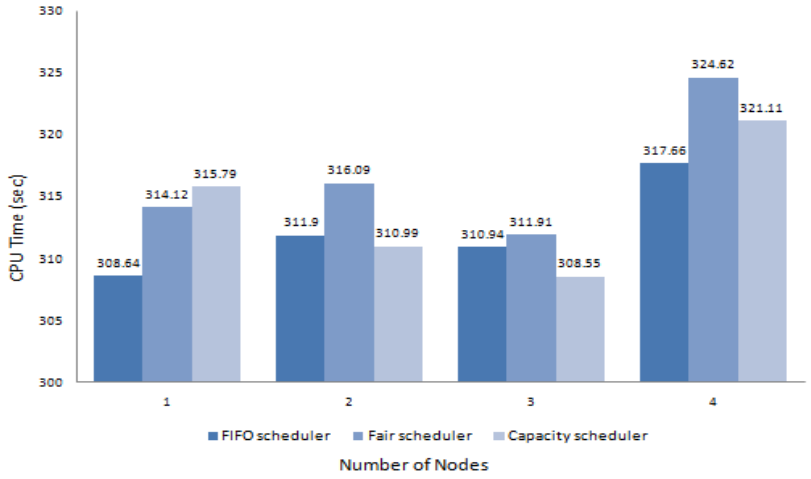

Fig. 7. Effect of the number of nodes on CPU Time for processing of 818.5 MB of data set by Wordcount job

In this experiment, the capacity provided to the queue is 100%. Therefore, the queue uses the entire capacity of the cluster. The job utilizes all the Map slots of the entire cluster, which improves the execution time of the job as well as improves the performance of the cluster. Another property Data locality rate helps to decrease the processing time of a job. The data locality rate of Capacity scheduler on 1, 2, 3, and 4 number of nodes is 100%. The data locality rate is the number of data local map tasks launched divided by the total number of map tasks launched. This means the data local Map tasks are almost same to the launched Map tasks. For example, when a job is executed on 3 nodes, the data local Map task were 14 and launched Map tasks were also 14. Thus, Data locality rate is 100%. The FIFO scheduler's processing time increases as the number of nodes increases because FIFO does not consider Data locality rate. If the data is not nearer to the computation node, the job has to wait for execution because of which the processing time increases.

As shown in Fig. 7, the performance of the Fair scheduler decreases when the number of nodes is 3. The reasons behind the degradation of performance can be the network cost, the data locality, and the time taken by a task to get launched on a node. The Data locality rate of the Fair scheduler on 1, 2, 3, and 4 number of nodes is 100%, 100%, 71%, and 81%, respectively. The data locality rate of the Capacity scheduler is 100%, 100%, 90 %, and 90 %, which is better than the Fair scheduler data locality rate. This observation shows that the Capacity scheduler performs better than the Fair scheduler.

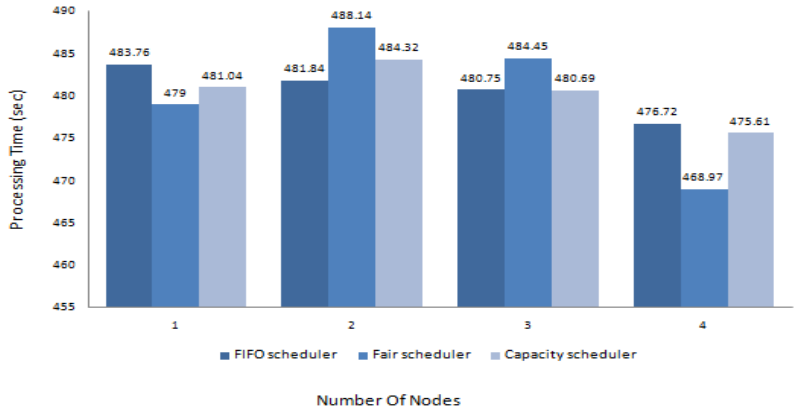

Fig. 8. Effect of the number of nodes on CPU Time for processing of 1.2 GB of data set by Wordcount job

The performance of the Capacity scheduler increases as the number of nodes increases. The overall performance of Capacity scheduler is good as compared to FIFO and Fair scheduler as shown in Fig. 8. The capacity scheduler takes less processing time as the number of nodes increases. The data locality rate of the Capacity scheduler is 100%, 100%, 97%, and 97% on 1, 2, 3, and 4 number of nodes, respectively. For the number of nodes 2 and 3, Fair scheduler takes more CPU time as compared to FIFO and Capacity schedulers. The reason behind this is network cost or most of the Map tasks may not be scheduled on the nodes where the data to be processed by them is located.

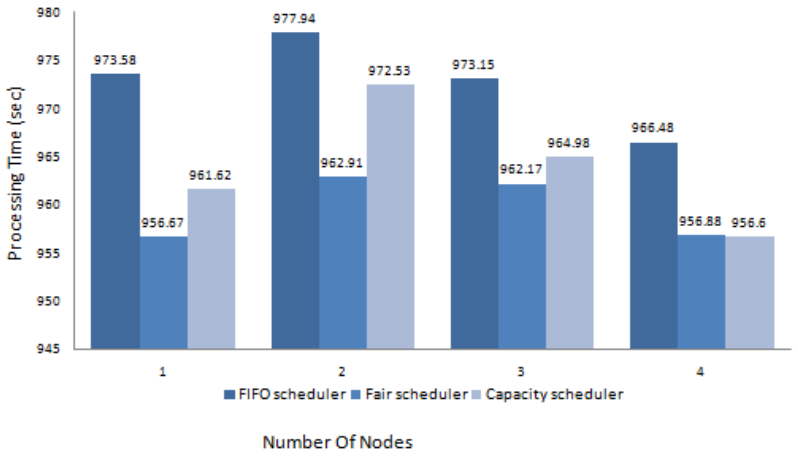

Fig. 9. Effect of the number of nodes on CPU Time for processing of 2.4 GB of data set by Wordcount job

Based on the CPU time, the performance of Capacity and Fair scheduler is better than FIFO scheduler for processing of 2.4 GB of data size and for a different number of nodes as shown in Fig. 9. As the number of nodes is increased the processing time of a job gets decreased. The parallelism is increased. Among the three schedulers, Fair scheduler takes less time to process a job. The minimum number of Maps

provided to Fair scheduler is 170 and the limit for a maximum number of jobs is 25. The number of Maps which a pool can use is more, so the processing time becomes less. The data locality rate of Fair scheduler is 100%, 81%, 97%, and 100% on 1, 2, 3, and 4 number of nodes, respectively. FIFO scheduler takes more CPU time as compared to Fair and Capacity schedulers. FIFO scheduler does not consider the data locality.

5. Conclusion and future work

In this article, we evaluated three schedulers available in Hadoop based on various parameters of the study. To carry out an experiment evaluation, we implemented our own Hadoop cluster having one master and four slaves. The article highlighted crucial information related to installation and troubleshooting, which might become useful to others who want to prepare their own Hadoop cluster. This article analysed the performance of Grep and Wordcount applications by varying data size (818.5 MB, 1.2 GB, and 2.4 GB) based on the performance metrics such as CPU time, Turnaround time, and Data processed per second.

From the experimental analysis, we can conclude that the performance of schedulers is dependent on data size, the number of nodes in the cluster, configurable parameters of schedulers, the number of jobs running in the cluster, and the type of applications. Different task schedulers have a different influence on the performance in different situations. The FIFO scheduler takes more time for processing a job as compared to the Fair and the Capacity scheduler while the Fair and the Capacity schedulers take almost same processing time. The Capacity scheduler takes more Turnaround time as compared to the Fair scheduler while the Fair scheduler takes less Turnaround time as compared to FIFO scheduler. As the Capacity scheduler uses the concept of queue capacity, it may happen that queues having less capacity take more time to process the job. Also, the time taken by a job to get launched for execution is higher. The Fair scheduler processes more data per second as compared to Capacity and FIFO scheduler. Among three schedulers, FIFO processes fewer data per second. We can also conclude that the data processing capability of Grep is more than that of running WordCount, when they both processes 2.4 GB of data set on the FIFO, the Fair scheduler, and the Capacity scheduler. Wordcount is a CPU intensive application while Grep is not CPU intensive application. So, Grep takes less time to process as compared to Wordcount.

The performance of a job in a cluster depends on Job configuration settings, cluster configuration, and input data. Increasing the number of nodes in the cluster increases the parallelism, which directly affects the performance. For further analysis of the performance of schedulers, more complex workload can be used and effect of HDFS block size can be observed. For improving the performance of Fair scheduler, additional configurable parameters such as the weight of pools and jobs can be used. For improving the performance of Capacity scheduler, the administrator can limit the number of concurrent jobs per queue and per user. Such analysis and experimental evaluation can be used to realize autonomic computing in which job scheduling policy at the master node can be adapted to the type of workload and changing cluster

environment. The job policy files or configuration files with adapted parameters can be generated dynamically using scripting languages such as python, perl, and Unix/Linux shell scripts, which are widely used for automation in Cloud Computing.

References

1. Bardhan, S., D. A. Menascé. The Anatomy of Mapreduce Jobs, Scheduling, and Performance Challenges. – In: Proc. of 2013 Conference of the Computer Measurement Group, 2013.
2. Apache Hadoop. Last accessed on 15 April 2016. **http://hadoop.apache.org**
3. Shilpa, M. K. Big Data Visualization Tool with Advancement of Challenges. – International Journal of Advanced Research in Computer Science and Software Engineering, Vol. **4**, 2014, No 3, pp. 665-668.
4. Davis, R. I., A. Burns. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. – ACM Computing Surveys (CSUR), ACM, Vol. **43**, 2011, No 4, p. 35.
5. Herodotou, H., S. Babu. Profiling, What-if Analysis, and Cost-Based Optimization of Mapreduce Programs. – In: Proc. of VLDB Endowment, 2011, pp. 1111-1122.
6. Wang, G., A. R. Butt, P. Pandey, K. Gupta. A Simulation Approach to Evaluating Design Decisions in Mapreduce Setups. – In: MASCOTS, 2009, pp. 1-11.
7. Zaharia, M., D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. – In: Proc. of 5th European Conference on Computer Systems, ACM, 2010, pp. 265-278.
8. Liu, S., J. Xu, Z. Liu, X. Liu. Evaluating Task Scheduling in Hadoop-Based Cloud Systems. – In: 2013 IEEE International Conference on Big Data, IEEE, 2013, pp. 47-53.
9. Gautam, J. V., H. B. Prajapati, V. K. Dabhi, S. Chaudhary. A Survey on Job Scheduling Algorithms in Big Data Processing. – In: 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), IEEE, 2015, pp. 1-11.
10. Zaharia, M. Hadoop Fair Scheduler Design Document, August 15 2009. Last accessed on 15 April 2016. **http://svn.apache.org/repos/asf/hadoop/common/branches/MAPREDUCE-233/src/contrib/fairscheduler /designdoc/fair_scheduler_design_doc.pdf.**
11. Wang, D., J. Chen, W. Zhao. A Task Scheduling Algorithm for Hadoop Platform. – Journal of Computers, Vol. **8**, 2013, No 4, pp. 929-936.
12. Gu, R., X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan, Y. Huang. Shadoop: Improving Mapreduce Performance by Optimizing Job Execution Mechanism in Hadoop Clusters. – Journal of Parallel and Distributed Computing, Vol. **74**, 2014, No 3, pp. 2166-2179.
13. Anjos, J. C., I. Carrera, W. Kolberg, A. L. Tibola, L. B. Arantes, C. R. Geyer. Mra++: Scheduling and Data Placement on Mapreduce for Heterogeneous Environments. – Future Generation Computer Systems, Vol. **42**, 2015, pp. 22-35.
14. Ling, X., Y. Yuan, D. Wang, J. Liu, J. Yang. Joint Scheduling of Mapreduce Jobs with Servers: Performance Bounds and Experiments. – Journal of Parallel and Distributed Computing, Elsevier, Vol. **90**, 2016, pp. 52-66.
15. Li, X., T. Jiang, R. Ruiz. Heuristics for Periodical Batch Job Scheduling in a Mapreduce Computing Framework. – Information Sciences, Elsevier, Vol. **326**, 2016, pp. 119-133.
16. Mashayekhy, L., M. M. Nejad, D. Grosu, Q. Zhang, W. Shi. Energy-Aware Scheduling of Mapreduce Jobs for Big Data Applications. – IEEE Transactions on Parallel and Distributed Systems, IEEE, Vol. **26**, 2015, No 10, pp. 2720-2733.
17. Tang, Z., J. Zhou, K. Li, R. Li. A Mapreduce Task Scheduling Algorithm for Deadline Constraints. – Cluster Computing, Springer, Vol. **16**, 2013, No 4, pp. 651-662.
18. Wang, Y., W. Shi. Budget-Driven Scheduling Algorithms for Batches of Mapreduce Jobs in Heterogeneous Clouds. – IEEE Transactions on Cloud Computing, IEEE, Vol. **2**, 2014, No 3, pp. 306-319.

19. L i u, Y., W. W e i. A Replication-Based Mechanism for Fault Tolerance in Mapreduce Framework. – Mathematical Problems in Engineering, Hindawi Publishing Corporation, Vol. **2015**, 2015.
20. C h e n, Q., M. G u o, Q. D e n g, L. Z h e n g, S. G u o, Y. S h e n. Hat: History-Based Auto-Tuning Mapreduce in Heterogeneous Environments. – The Journal of Supercomputing, Springer, Vol. **64**, 2013, No 3, pp. 1038-1054.
21. G u n a r a t h n e, T., B. Z h a n g, T.-L. W u, J. Q i u. Scalable Parallel Computing on Clouds Using Twister4azure Iterative Mapreduce. – Future Generation Computer Systems, Elsevier, Vol. **29**, 2013, No 4, pp. 1035-1048.
22. S u n, M., H. Z h u a n g, C. L i, K. L u, X. Z h o u. Scheduling Algorithm Based on Prefetching in Mapreduce Clusters. – Applied Soft Computing, Elsevier, Vol. **38**, 2016, pp. 1109-1118.
23. S e h r i s h, S., G. M a c k e y, P. S h a n g, J. W a n g, J. B e n t. Supporting hpc Analytics Applications with Access Patterns Using Data Restructuring and Data-Centric Scheduling Techniques in Mapreduce. – IEEE Transactions on Parallel and Distributed Systems, IEEE, Vol. **24**, 2013, No 1, pp. 158-169.
24. W a n g, L., J. T a o, R. R a n j a n, H. M a r t e n, A. S t r e i t, J. C h e n, D. C h e n. G-Hadoop: Mapreduce Across Distributed Data Centers for Data-Intensive Computing. – Future Generation Computer Systems, Elsevier, Vol. **29**, 2013, No 3, pp. 739-750.
25. T i w a r i, N., S. S a r k a r, U. B e l l u r, M. I n d r a w a n. Classification Framework of Mapreduce Scheduling Algorithms. – ACM Computing Surveys (CSUR), ACM, Vol. **47**, 2015, No 3, p. 49.
26. L i, F., B. C. O o i, M. T. Ö z s u, S. W u. Distributed Data Management Using Mapreduce. – ACM Computing Surveys (CSUR), ACM, Vol. **46**, 2014, No 3, p. 31.
27. L e e, K.-H., Y.-J. L e e, H. C h o i, Y. D. C h u n g, B. M o o n. Parallel Data Processing with Mapreduce: A Survey. – AcM sIGMoD Record, ACM, Vol. **40**, 2012, No 4, pp. 11-20.
28. I n a c i o, E. C., M. A. D a n t a s. A Survey into Performance and Energy Efficiency in hpc, Cloud and Big Data Environments. – International Journal of Networking and Virtual Organisations, Inderscience Publishers, Vol. **14**, 2014, No 4, pp. 299-318.
29. A l t h e b y a n, Q., Y. J a r a r w e h, Q. Y a s e e n, O. A l Q u d a h, M. A l - A y y o u b. Evaluating Map Reduce Tasks Scheduling Algorithms over Cloud Computing Infrastructure. – Concurrency and Computation: Practice and Experience, Wiley Online Library, Vol. **27**, 2015, No 18, pp. 5686-5699.
30. J i a, Z., R. Z h o u, C. Z h u, L. W a n g, W. G a o, Y. S h i, J. Z h a n, L. Z h a n g. The Implications of Diverse Applications and Scalable Data Sets in Benchmarking Big Data Systems. – In: Specifying Big Data Benchmarks, Springer, 2014, pp. 44-59.
31. H e, C., Y. L u, D. S w a n s o n. Matchmaking: A New Mapreduce Scheduling Technique. – In: 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2011, pp. 40-47.