MOMC: Multi-Objective and Multi-Constrained Scheduling Algorithm of Many Tasks in Hadoop

Cristiana Voicu, Florin Pop, Ciprian Dobre
University *Politehnica* of Bucharest, Romania.
Emails: cristiana.voicu@hpc.pub.ro, florin.pop@cs.pub.ro, ciprian.dobre@cs.pub.ro

Fatos Xhafa
Universitat Politècnica de Catalunya
Barcelona, Spain.
Email: fatos@lsi.upc.edu

***Abstract*—Even though scheduling in a distributed system was debated for many years, the platforms and the job types are changing everyday. This is why we need special algorithms based on new applications requirements, especially when a application is deployed in a Cloud environment. One of the most important framework used for large-scale data processing in Clouds is Hadoop and its extensions. Hadoop framework comes with default algorithms like FIFO, Fair Scheduler or Capacity Scheduler, and Hadoop on Demand. These scheduling algorithms are focused on a different and single constraint. It is hard to satisfy multiple constraints and to have a lot of objectives in the same time. After summarizing the most common schedulers, showing the need of each one in the moment it appeared on the market, this paper presents MOMC, a multi-objective and multi-constrained scheduling algorithm of many tasks in Hadoop. MOMC implementation focuses on two objectives: avoiding resource contention and having an optimal workload of the cluster, and two constraints: deadline and budget. To compare the algorithms based on different metrics, we use Scheduling Load Simulator, which is integrated in Hadoop framework and helps the developers to spend less time on testing. As killer application that generate many tasks we have chosen processing task for the Million Song Dataset, which is a set of data contains metadata for one million commercially-available songs.**

Keywords*—*Task Scheduling, Hadoop, MapReduce, Big Data, Cloud Computing.

I. INTRODUCTION

Nowadays every enterprise information system, such as in intelligent decision support, business analytics, knowledge discovery, cloud storage, distributed computing, etc. is working with a huge amount of data. Since the time and the hardware are two valuable resources, it is very important to schedule the data processing very effective. This is way the algorithms behind the scheduler are an important topic for parallel and distributed system [1].

Hadoop is one of the most known system used to process huge amount of data, an environment that offers distributed high-performance processing of data. The framework is an open source project and it became very big because many companies have contributed to it. Since every company collects different kind of data and processes it in a different way, every one has to implement its own scheduler. The most known methods for scheduling in Hadoop are: FIFO scheduler, Fair scheduler and Capacity scheduler [2], [3]. Besides these well known algorithms, there are other schedulers with different features: scheduling to meet deadline [4], the total budget, data dependencies, computational dependency, etc. [5].

The goal of this paper is that, based on the existing methods to schedule, to design and implement a new algorithm which should take into account multiple constraints, set by the user and it is used for many tasks computing in a Hadoop environment. The algorithm is called MOMC - Multi-Objective and Multi-Constrained Scheduling Algorithm of Many Tasks in Hadoop. Also, MOMC focuses on different objectives of scheduling (budget cuts, minimizing total execution time, load balancing of nodes, energy efficiency) [6] [7].

Deploying a scheduling algorithm in a real Hadoop cluster means to analyze a lot of information and it is really hard to obtain the big picture. This is why a simulator was implemented as part of Hadoop framework. This simulator takes the workload from the jobs history and shows in real time various graphics based on different metrics. So, the evaluation of MOMC algorithm presented in this paper focuses on Scheduling Load Simulator (SLS) [8], [9]. The scheduling solutions already implemented are used to compare the results.

The killer application chosen to evaluate MOMC algorithm parses the million song Dataset and obtains various results and statistics. The Million Song Dataset [10] contains metadata for one million commercially-available songs.

The rest of the paper is structured as follow. The second section presents the existing schedulers in Hadoop: FIFO, Fair Scheduler, Capacity Scheduler and HOD Scheduler and take a short overview on existing solution for multi-objective and/or multi-constrains scheduling. The next section presents details of our new algorithm, MOMC: the purpose, pseudo-code of some parts of it and the mathematical formulas. MOMC algorithm is based on two objectives: avoiding resource contention and having an optimal workload of the cluster,and two constraints: deadline and budget. In section six, we present the experimental results, based on various metrics. One subsection presents the workload and how to get the jobs history from Hadoop; the other section shows multiple graphics with the comparisons between our own algorithm, FIFO and Fair algorithm. The paper ends with further directions and conclusions.

II. RELATED WORK

Hadoop is a reliable and scalable system from one to many servers, each of them having local storage and computation. Hadoop offers the big advantage of a distributed system: high availability, because each failure is detected and handled at the application layer. Hadoop is used for various purposes which need high performance processing of data: Facebook tasks execution with Corona, human genome decoding, e-science simulations, etc.

A long period of time, Hadoop had just one scheduler, used by the JobTracker node. This was enough for that moment, but later on it revealed to be inflexible for different constrains. The next step was to implement an independent and pluggable scheduler. The big advantage of this new component was that it allows implementing a new algorithm for the scheduler that can be easily used in practice. In this case, the right algorithm can be used to optimize the jobs. We take a short overview of exiting scheduling methods in Hadoop to highlight the main constraints and optimization objectives considered.

FiFO scheduler is the method implemented inside the JobTracker, before creating the pluggable scheduler. It uses a queue with jobs, so the oldest task is assign first to a slave. It has a simple implementation, but it does not take into account any other constraint.

Fair scheduler, developed by Facebook, the main idea of this type of scheduler is to give to each job equal resources. It is called *fair* because it helps the small tasks to run in the same time with another job that occupies more the CPU. The sadvantage for this scheduler is that the all system has greater responsiveness to the variety of job types submitted. In the background, a set of pools is created and each of them has a set of shared resources. This helps to balance the resources across the jobs that will be plan by the scheduler. This method of scheduling permits to modify the configuration, to set how the sharing between the pools is done. Another constrained implemented by this type of scheduler is the number of jobs, because in case there are too many jobs, it is possible to have congestion, which will aggravate the computation. Users will receive the same amount of resources, because each one has its individual pool. This assures the fairness for the system, because each job will be independent. If the system is not used, users will share the amount of resources which are free, splitting the resources.

Capacity scheduler, developed by Yahoo, capacity scheduler has some features of the fair scheduler, but it has some distinct characteristics. In the next words, we expose the features of capacity scheduler, in comparison with fair scheduler. It was designed for big clusters, with many servers, with various usages. Unlike fair scheduler which uses pools, capacity scheduler uses queues. Each queue can have a different number of tasks (map or reduce tasks) and this number can be set in the configuration files. Also, a queue has a guaranteed capacity. Like fair scheduler, if the resources are not used by a queue, those are split to the ones that need them. The improvement for this method of scheduling was the possibility to set priorities for the jobs inside the queue. If a job has a higher priority, this means that it will get first the resources. To prioritize even better, they can use the preemption: stop the low priority tasks to let those with high priority to run. This feature is not implemented yet. Each queue can be administrated by a person or organization. Because of this, it needs a system to grant different types of access. Capacity scheduler permits to modify if the user can submit jobs, view or modify them.

HOD Scheduler is an easy to use tool, which allows to easy setup and use Hadoop. MapReduce and HDFS are totally independent for a cluster of nodes. In general it is used to share the physical cluster to deploy their Hadoop versions. To allocate a node, it uses Torque resource manager [11], and on each node two daemons are started: Hadoop MapReduce and HDFS [12]–[15].

YARN or MRv2 (MapReduce 2.0) comes with a big improvement because it splits in different components the Hadoop job tracker, the resource manager and the functionality used to schedule the jobs [16].

New extensions are oriented on optimizations to the MapReduce job and task execution mechanisms [17]: optimization of setup and cleanup tasks to reduce the time cost during the initialization and termination stages of the job; and an instant messaging communication mechanism for accelerating performance-sensitive task scheduling and execution. The security aspects, which are not considered in this paper, are presented in [18].

III. MOMC Scheduling Algorithm

The focus of this paper is on MOMC algorithm that could satisfy both constraints and objectives. Since we wanted to obtain a scheduler for Hadoop, we did not take into account how the data is transferred from outside. We considered that the input is already placed in HDFS and are available at run time for any job.

We consider the most important objectives for such an environment and our conclusion was that we need to satisfy the following two: avoiding resource contention and having an optimal workload of the cluster. Regarding the constraints, these can vary, but since time and money are the most important factors, we finally considered deadline and budget. The difference between the constraints and the objectives is that the first are focused on the framework internals, but the second should be set by the user.

A. Constraints

An issue usually met for a scheduler is the heterogeneity. It is hard to take into account the differences between the tasks or resources. In order to avoid to prevent the conflicts between the resources, we tried to obtain as many information for each job, like CPU, IO or memory.

Regarding the heterogeneity, in our scheduling algorithm we have tried to satisfy this, by finding to every step, the best match between the job and the resource. But the best advantage is that we wanted to have the big picture of the all cluster, finding the best fit between the all jobs that need to run at a certain moment and the available resources.

B. Objectives

The objectives are more related to what the user wants to obtain. In this way, the user should specify the values for the deadline and the budget in the scheduler configuration file. If those are not specified, the default values will be used. Also, the information about map and reduce tasks is needed, as well as information about the nodes. We need to know CPU, IO and memory, and those should be specified in the configuration files.

C. The MOMC Model

Let consider a job J that should run on a Hadoop cluster. Each job J has a known number of map tasks and reduce tasks, t_m, respectively t_r. For each job, we know from the beginning the exact amount of data that should be processed. We call this map input, I, because the data processed by reducers is not known. We can approximate it using the previous jobs. We call r the ratio that filter the amount of input data.

Algorithm 1 MOMC Scheduling Algorithm

```
1:  for each $workernode \in workerNodes$ do
2:      $maxService \leftarrow workernode.service$;
3:      for each $assignedjob \in workernode.assignedJobs$ do
4:          $newAssignedJobs \leftarrow Workernode.assignedJobs - assignedjob$;
5:          for each $job \in jobQueue$ do
6:              $newAssignedJobs \leftarrow newAssignedJobs \bigcup job$;
7:              if $service(newAssignedJobs) > service(maxService)$ then
8:                  $maxService \leftarrow newAssignedJobs$;
9:              end if
10:         end for
11:     end for
12:     $workernode.assignedJobs \leftarrow newAssignedJobs$;
13: end for
```

Each job has an arrival time, A and other two identifiers to deal with constraints: deadline - D and budget - B available for the execution. So, each job can be identified as a tuple of (A, I, D, B). Each job J has a specific number of slots assigned, some of them for maps s_m and others for reducers s_r.

D. The MOMC Scheduling Algorithm

The proposed MOMC algorithm follow the best usage jobs-resources strategy. Algorithm 1 presents the main steps of MOMC. The $service$ function should return if there are enough mappers and enough reducers in order to finish the job in the specified budget and until the deadline ends. To simplify, we agreed to return 0 or 1 (0 when the resource is not good for the job, 1 when everything is OK). The Algorithm 1 verifies each assignment between jobs and resources, and then tell which assignment is better by the sum of each $service$ result.

To compute the $service$, we need to find out if s_m and s_r can be provided by a specific resource. For this, we define $timeCost_m$ and $timeCost_r$ as the time cost to process the amount of data by mappers, respectively by reducers. Also, we define $budgetCost_m$ and $budgetCost_r$, the budget cost to process the amount of data by mappers, respectively by reducers. Besides those costs, we need to take into account the cost paid when the data is not on the node where the reducers run. So, we should define the time cost for data, $timeCost_d$, and budget cost for data, $budgetCost_d$. Let us consider $start_m$ and $start_r$, the start time for the mappers, respectively for the reducers. To compute the number of map slots needed by the job J, we need to know the maximum value of $start_r$, $start_r^{max}$.

These are the formulas used to compute s_m^{min} and s_r^{min}, the minimum number of map slots, respectively of reduce slots needed to finish the computation of job J, based on time:

$$s_m^{min} = \frac{timeCost_m}{start_r^{max} - start_m} \quad (1)$$

$$s_r^{min} = \frac{rtimeCost_r}{A + D - rtimeCost_d - start_r} \quad (2)$$

Based on the time processing cost, we can conclude that the budget is the multiplication between time and resources. If the user budget is less that what a specific resource can provide, the $service$ will be 1.

E. Scheduler Implementation in Hadoop

Since JobTracker is an independent component, it is not so hard to implement a new algorithm in Hadoop. The scheduler class should extend TaskScheduler, along with the properties and the methods). When the implementation is done, the scheduler needs to be plug-in. In the configuration files of MapReduce there is a variable that indicates the used algorithm. By default, it uses the Fair Scheduler, but when you want to use your own implementation, you just have to modify the configuration.

IV. Million song Dataset Application for Hadoop

In order to test each Hadoop scheduling algorithm and the new one (MOMC), we need a killer application written for Hadoop. The application should analyze a big amount of data, so we have chosen the Million Song Dataset [10], [19]. This set of data contains metadata for one million commercially-available songs.

Based on the information provided by the million song dataset, we implemented various tasks that analyze it, in order to obtain statistical information. We inspected the MSD using Hadoop and we done the following tasks: get the evolution of the loudness war (average loudness as a function of the year), the same task for tempo, sort the genres by the number of song in the dataset, by the average loudness, by the average tempo. Another task was to show the evolution of the relevant characteristics side-by-side for major and minor keys. Minor keys have traditionally been reserved for songs which seek to convey sadness or melancholy. As such, we can reasonably expect minor keys to be correlated with low tempos and reduced loudness.

The application is written using Hadoop MapReduce framework and it is implemented in Java, like the framework. We shortly explain what is happening with the job when it runs in the background. MapReduce, as the name says, has two important types of tasks: map and reduce. The data received from input is divided in chunks, so a map task receives just one part of the entire data. Many map tasks run in parallel. The output from all the map tasks is sorted and redirected to reducers. The data is stored in a file system, so Hadoop uses HDFS (Hadoop Distributed File System) for this part. The scheduler is in charged to plan how the jobs are assigned to map or reducers.

In practice, a node is in charged with computation and storage. In this way, it is desired to schedule the task on the node with the date it needs, in order to reduce the traffic in the network. But for this to happen, the scheduler needs an improved algorithm. For MSD application, we have implemented map and reduce functions using the appropriate interfaces.

V. Setup and Implementation Details

This section presents all the setup steps needed to develop and then to test the Hadoop scheduler. We start with the configurations needed for Hadoop framework, followed by the scheduler load simulator setup. The last section explain our first Hadoop application; its purpose was to understand the paradigm behind the scene, the nature of the jobs and of the tasks.

A. Hadoop Setup

We developed the scheduler on a machines with Ubuntu 13.10, a 64 bits architecture. Based of this, we faced some issues with some libraries and the solution was to compile Hadoop source code on our system. Before this, you need to instal maven, libssl-dev, cmake and protobuf v2.5.0.

B. HDFS configuration

In `core-site.xml` file, you should specify the value of `fs.default.name`, meaning the host and the port where the hadoop file system will run. Besides this, you have to add the value of `dfs.replication` in `hdfs-site.xml` configuration file.

C. YARN configuration

In `mapred-site.xml`, there are multiple variables that should be filled (see Figure 1).

```
<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>localhost:9001</value>
    </property>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapreduce.cluster.local.dir</name>
        <value></value>
        <final>true</final>
    </property>
    <property>
        <name>mapreduce.jobhistory.address</name>
        <value>localhost:10020</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>localhost:19888</value>
    </property>
</configuration>
```

Fig. 1. mapred-site.xml configuration file

In Figure 2 you can see the configuration needed to be added to `yarn-site.xml` file.

D. SLS Setup

The simulator has its own configuration file `sls-runner.xml`, which be placed in the directory `/etc/hadoop`. In this file you have to mention multiple variables, like the entire simulator memory or the one for each container, the number of vCore, and so on. You can find a sample for this file in SLS source code. This is placed in the directory `share/hadoop/tools/sls/sample-conf`.

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.resourcemanager.scheduler.class</name>
        <!--<value>org.apache.hadoop.yarn.server.resourcemanager
                .scheduler.fair.FairScheduler</value>-->
        <value>org.apache.hadoop.yarn.server.resourcemanager
                .scheduler.fifo.FifoScheduler</value>
        <!--<value>org.apache.hadoop.yarn.server.resourcemanager
                .scheduler.capacity.CapacityScheduler</value> -->
    </property>
</configuration>
```

Fig. 2. yarn-site.xml configuration file

E. Scheduling Load Simulator

When a new scheduling algorithm is implemented, it is very hard to deploy it in real clusters. What if the developers will have a way to just gather the output data of the cluster jobs and then analyze them in a simulator? This is way SLS (Scheduling Load Simulator) was designed and implemented. Using it is easier to deploy various and different features for a scheduler, because you can use only one machine to deploy it. In this way, the SLS saves a lot of time and, of course money, because the deployment of a new scheduler is cost and time consuming.

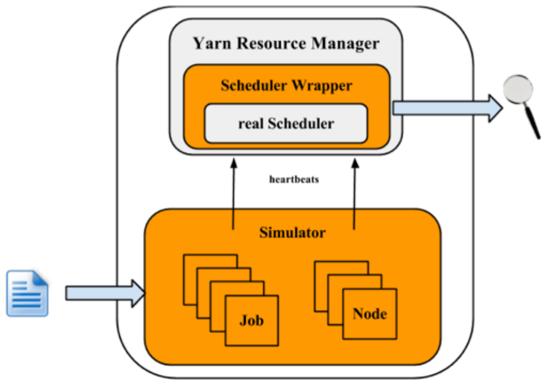


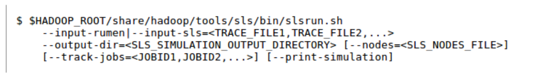
Fig. 3. SLS architecture

Figure 3 shows the architecture of SLS. As we previously mentioned, everything runs on a single machine. This means that all YARN components should run on the same machine, in the same JVM, without network component. Since one of the main component of the Resource Manager is the Scheduler, the major changes needed for SLS are in there. Basically, SLS acts like a wrapper for the principal scheduler.As you can see in the figure, the orange parts are those implmented for SLS. In the Simulator part, SLS simulates each node manager and each application manager. In this section, the focus was more on the SLS architecture, but in the next section we describe more the input data for it.

F. Rumen

Figure 4 shows how to run the SLS. It takes as input some Rumen trace and it puts the results in a mentioned directory.

The path to the script that runs the simulator from the hadoop install directory is `share/hadoop/tools/sls/bin`.



```
$ $HADOOP_ROOT/share/hadoop/tools/sls/bin/slsrun.sh
    --input-rumen|--input-sls=<TRACE_FILE1,TRACE_FILE2,...>
    --output-dir=<SLS_SIMULATION_OUTPUT_DIRECTORY> [--nodes=<SLS_NODES_FILE>]
    [--track-jobs=<JOBID1,JOBID2,...>] [--print-simulation]
```

Fig. 4. Run SLS

When jobs are running to perform map or reduce tasks, the all history is kept by the JobHistory daemon. But this history is not entire in order to be shown based on the sls metrics, so the simulator use a different input. Rumen tool is used to transform and to append information to jobs history and the output of it will be a json file. This json file will be further used as input for the simulator.

There are two steps in the transformation process: Trace-Builder and Folder. The first one is meant to process the data in order to have a better format. The second one is used to add additional information based on some statistical calculations. These two steps are performed using two command that can be found in Rumen tools directory in Hadoop framework. To find any other information about how these commands should be used, the parameters and the options, please visit [20].

G. *SLS Metrics*

The programming language used in Hadoop source code is Java and the most used library to measure the performance in Java is Metrics [21]. The scheduling load simulator use this library to evaluate the results of an algorithm. In this section I will explain the metrics used in SLS, metrics that be used to show the results of our own algorithm compared to most used algorithms. The main focus of these metrics is on time cost and how the resources are used.

The metrics are the following:

- Applications and containers: how many applications and containers are running during the time;
- Resources: how many resources are used and how many resources are left;
- Resources per queue: the used and left resources per each queue;
- Time: there are multiple operations done by each scheduler, like adding, removing, and updating a node, adding and removing and application, so the time for each operation means a lot;
- Memory: useful to know memory the simulator used.

VI. Experimental Scenarios

This section shows the tests done on the new scheduling algorithm. We compare it with other two basic algorithms: FIFO Scheduler and Fair Scheduler. We chose to compare with these two schedulers because they are already implemented in the Hadoop framework, therefore it did not require additional work. In order to use SLS, we need two input files: the json file with the workload and the json file with the topology. The first section will explain step by step how we get the workload of 50 MDS applications, which describe the usage of proposed applications by many people making during a specific period of time (2 hours).

A. *Experimental Workload*

In order to run the applications, you have to start all the hadoop java processes. Before this, you should format the `namenode` (this step is done just once). In order to start distributed file system daemon you should use `start-dfs.sh` script. This will start the `namenode` and one `datanode`.

We run 50 jobs on hadoop and then we used Rumen to obtain the workload in the format needed by the simulator. For this, we used the following command, which calls the TraceBuilde, takes as input the history and generates the jobs traces and the topology used. The last one is not needed because we create our own topology with 3 nodes.

If the name node is in safe mode and it does not let you to do the tasks, you can use the following command to leave it: `bin/hadoop dfsadmin -safemode leave`.

B. *Experimental Results*

We used MDS applications with different input data. We have four types of input, classified by its size: small, medium, big and extra big.

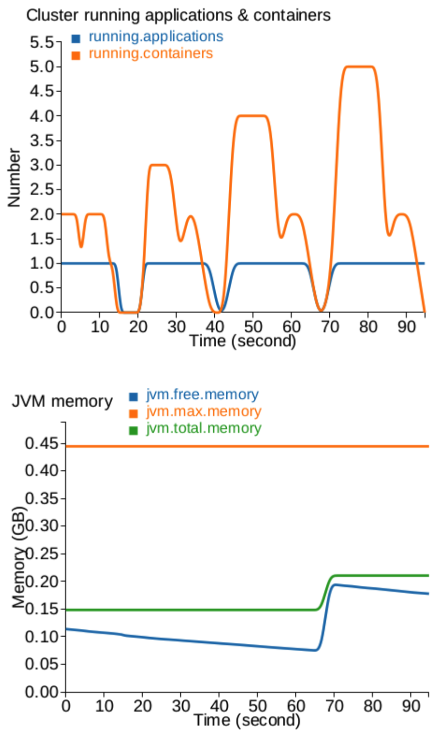


Fig. 5. FIFO scheduler applications and memory

To obtain simple results to better observe the differences between the algorithm we tested them only with four jobs. Each job will have different kind of input. The figures 5, 6, 7 represent the number of the applications and containers and the JVM memory for each algorithm, FIFO, Fair and respectively, our own algorithm, multi-objective and multi-constrained (MOMC).

The comparisons should be made on each column: the column on the left shows the difference regarding the number of applications and containers. The jobs are submitted for all of them in the same order: job with the small input first, then medium input, big and extra big. In case of 5, the jobs are executed in the same ordered as they are submitted. In case

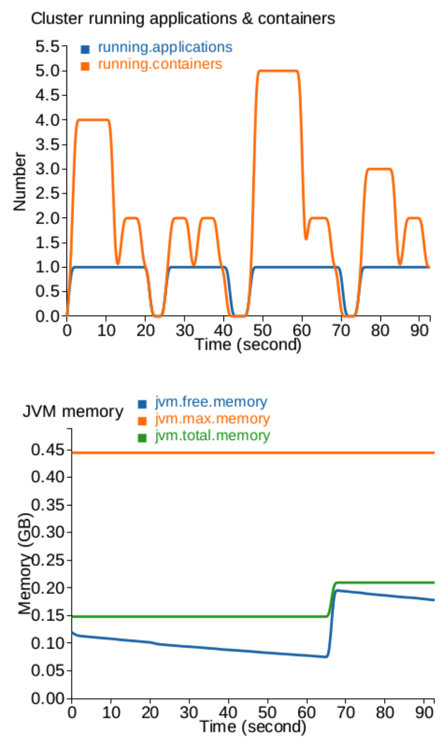


Fig. 6. Fair scheduler applications and memory

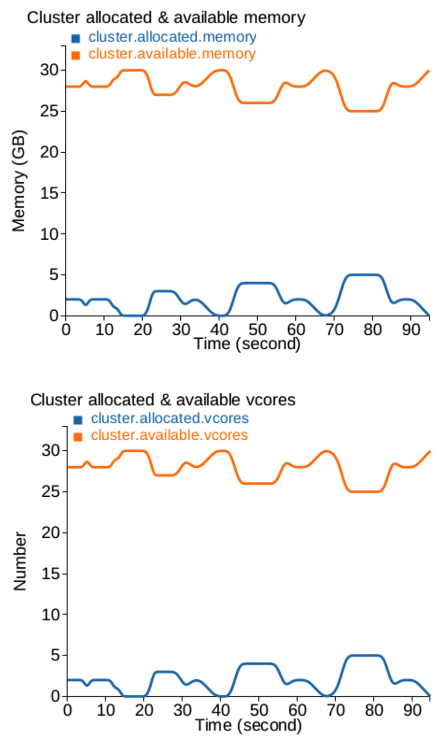


Fig. 8. FIFO scheduler cluster memory/vcores

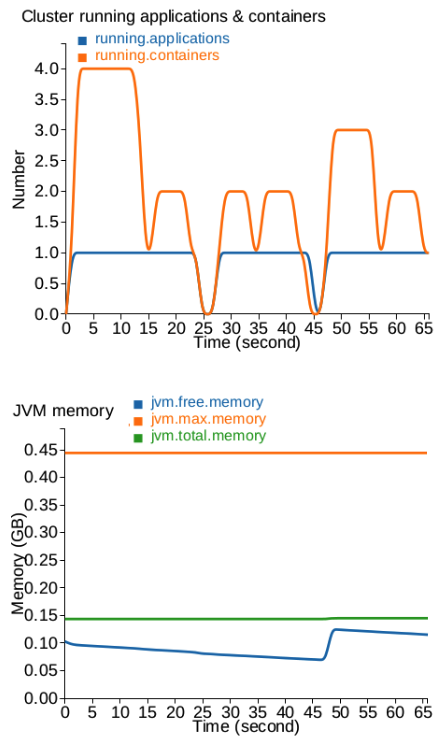


Fig. 7. Multi-objective and multi-constraint scheduler applications and memory

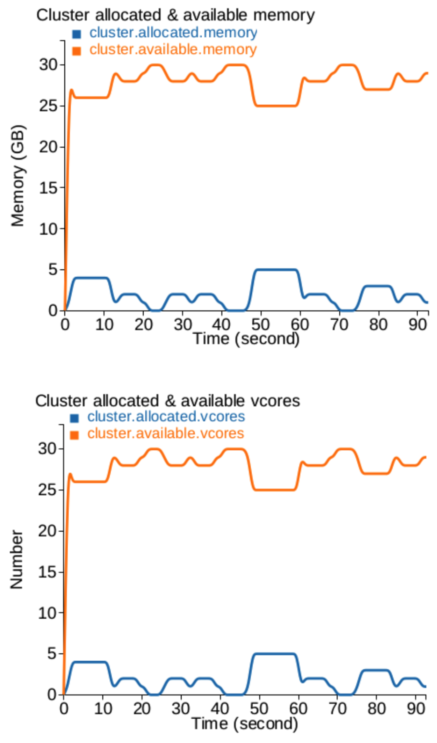
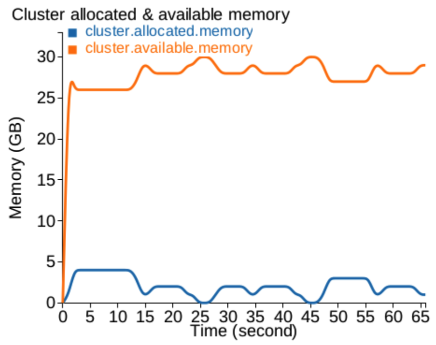


Fig. 9. Fair scheduler cluster memory/vcores

of 6, the scheduler rearranges the jobs because all of them should have access to the resource, even if one has more input. Regarding our MOMC algorithm, the most important conclusion is that the biggest job do not have permission to run. This is because the input is too big, so the deadline and the budget will exceed the expected ones.

The right column shows the amount of memory used by the entire simulator. When a job uses more memory for map or reduce task, the simulator memory will increase. We can clearly see that FIFO and FAIR used more memory because they run the job with extra big input. This is another aspect



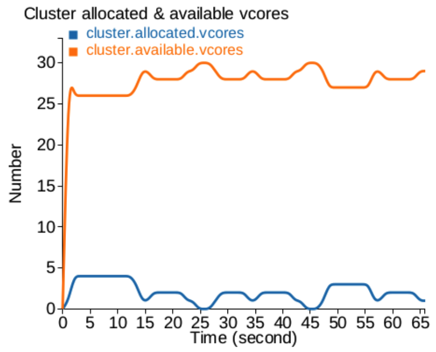


Fig. 10. Multi-objective and multi-constraint scheduler cluster memory/vcores

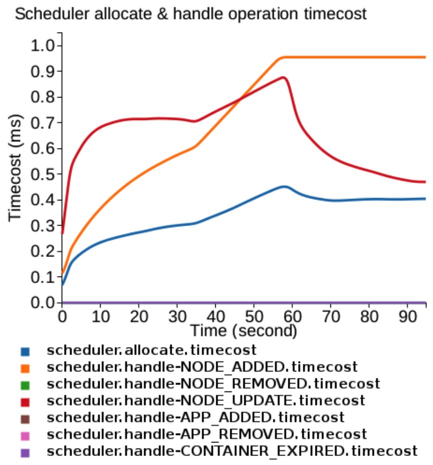


Fig. 11. FIFO scheduler operations timecost

which shows that MOMC do not run the tasks which exceeds the deadline, the budget or both of them.

Figures 8, 9 and 10 contain similar results: in the left we can see how much memory is allocated in the cluster and how much is available, and in the right column we can see how many vcores are allocated in the cluster and how many are available. These two types of graphics are similar due to the dependency between the memory and the vcores. Also, they are similar with the first type of metric.

The last type of metric is very important because it presents the time cost for each operation: allocate, node added, node removed, node update, application added, application removed, container expired. As you can see in Figures 11, 12 and 13, during the tests only three operations were done: allocate, node

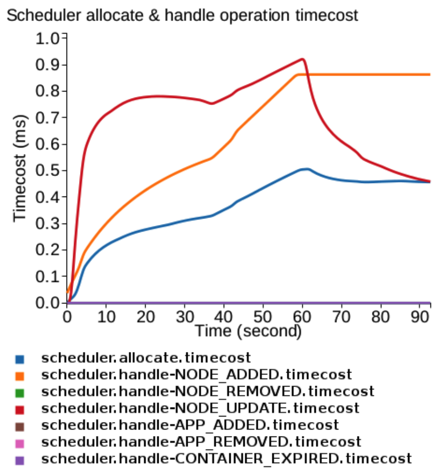


Fig. 12. Fair scheduler operations timecost

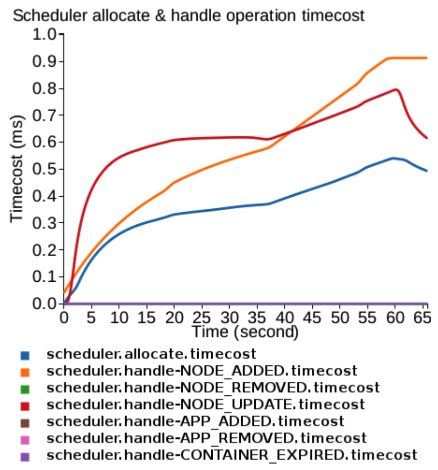


Fig. 13. Multi-objective and multi-constraint scheduler operations timecost

added and node removed. During the entire execution time, the cost for these operations is lower in MOMC case.

VII. Conclusions

This paper proposed a new algorithm named MOMC, which focused on two objectives: one is to find a multi-constrained and multi-objective algorithm and another one is Hadoop framework. MOMC covers the most relevant constraints for Hadoop (avoiding resource contention and having an optimal workload of the cluster) and relevant objectives (deadline and budget). Based on the configuration, our new scheduler proved to execute only the correct jobs. This kind of scheduler is useful when you have cost and time limitations. The killer application chosen to evaluate MOMC algorithm parses the million song Dataset and obtains various results and statistics. It follows the mentioned limitations.

We considered the following model: all MapReduce jobs are independent (Iterated MapReduce used for workflows will be considered as future work), there are no failure of nodes before/during scheduling computation, the scheduling decision is taken only based on current knowledge (learning from past iterations to swiftly compute the scheduling in next generations will be considered as future work), the optimization objectives are not considered to be "contradicting" (we will consider also this aspect as future work). Another important aspect are the metrics used to evaluate MOMC algorithm. We used the existing ones implemented for the other basics algorithms, which help us to compare between them. In the future, we are planning to research on this, to find relevant metrics for our own algorithm. In this way, we can better show the importance of having a multi-objective and multi-constraint scheduling algorithm. Other objective for MOMC extension will be the support of different workflow models for enterprise information systems in the Cloud.

Acknowledgment

The research presented in this paper is supported by projects: "*SideSTEP - Scheduling Methods for Dynamic Distributed Systems: a self-* approach*", ID: PN-II-CT-RO-FR-2012-1-0084; *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *MobiWay*: Mobility Beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow - PN-II-PT-PCCA-2013-4-0321; *clueFarm*: Information system based on cloud services accessible through mobile devices, to increase product quality and business development farms - PN-II-PT-PCCA-2013-4-0870.

This work was also partially supported by COMMAS Project "Computational Models and Methods for Massive Structured Data" (TIN2013-46181-C2-1-R).

References

[1] A. J. Page, T. M. Keane, and T. J. Naughton, "Scheduling in a dynamic heterogeneous distributed system using estimation error," *J. Parallel Distrib. Comput.*, vol. 68, no. 11, pp. 1452–1462, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2008.07.004

[2] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 29–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855744

[3] Z. Guo and G. Fox, "Improving mapreduce performance in heterogeneous network environments and resource utilization," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, ser. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 714–716. [Online]. Available: http://dx.doi.org/10.1109/CCGrid.2012.12

[4] K. Kc and K. Anyanwu, "Scheduling hadoop jobs to meet deadlines," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 388–392. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.97

[5] F. Zhang, J. Cao, K. Li, S. U. Khan, and K. Hwang, "Multi-objective scheduling of many tasks in cloud platforms," *Future Generation Computer Systems*, vol. 37, no. 0, pp. 309 – 320, 2014, special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13001854

[6] Y. Xia, L. WANG, Q. ZHAO, and G. ZHANG, "Research on job scheduling algorithm in hadoop," *Journal of Computational Information Systems*, vol. 7, no. 16, pp. 5769–5775, 2011.

[7] A. Rasooli and D. G. Down, "A hybrid scheduling approach for scalable heterogeneous hadoop systems," in *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, ser. SCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1284–1291. [Online]. Available: http://dx.doi.org/10.1109/SC.Companion.2012.155

[8] J. Dai, J. Huang, S. Huang, B. Huang, and Y. Liu, "Hitune: Dataflow-based performance analysis for big data cloud," in *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 24–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=2170444.2170468

[9] S. B. Joshi, "Apache hadoop performance-tuning methodologies and best practices," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12. New York, NY, USA: ACM, 2012, pp. 241–242. [Online]. Available: http://doi.acm.org/10.1145/2188286.2188323

[10] "Million song dataset," http://labrosa.ee.columbia.edu/millionsong/, accessed: 2014-06-01.

[11] G. Staples, "Torque resource manager," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188464

[12] X. Wang and J. Su, "Research of distributed data store based on hdfs," in *Proceedings of the 2013 International Conference on Computational and Information Sciences*, ser. ICCIS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1457–1459. [Online]. Available: http://dx.doi.org/10.1109/ICCIS.2013.384

[13] L. Wang, J. Tao, Y. Ma, S. Khan, J. Kolodziej, and D. Chen, "Software design and implementation for mapreduce across distributed data centers," *Applied Mathematics and Information Sciences*, vol. 7, no. 1 L, pp. 85–90, 2013, cited By (since 1996)2.

[14] L. Wang, D. Chen, R. Ranjan, S. Khan, J. Kolodziej, and J. Wang, "Parallel processing of massive eeg data with mapreduce," in *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, 2012, pp. 164–171, cited By (since 1996)1.

[15] L. Wang, J. Tao, H. Marten, A. Streit, S. Khan, J. Kolodziej, and D. Chen, "Mapreduce across distributed clusters for data-intensive applications," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2012*, 2012, pp. 2004–2011, cited By (since 1996)2.

[16] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633

[17] R. Gu, X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan, and Y. Huang, "Shadoop: Improving mapreduce performance by optimizing job execution mechanism in hadoop clusters," *J. Parallel Distrib. Comput.*, vol. 74, no. 3, pp. 2166–2179, Mar. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2013.10.003

[18] J. Zhao, L. Wang, J. Tao, J. Chen, W. Sun, R. Ranjan, J. Kolodziej, A. Streit, and D. Georgakopoulos, "A security framework in g-hadoop for big data computing across distributed cloud data centres," *Journal of Computer and System Sciences*, vol. 80, no. 5, pp. 994–1007, 2014, cited By (since 1996)0.

[19] B. McFee, T. Bertin-Mahieux, D. P. Ellis, and G. R. Lanckriet, "The million song dataset challenge," in *Proceedings of the 21st International Conference Companion on World Wide Web*, ser. WWW '12 Companion. New York, NY, USA: ACM, 2012, pp. 909–916. [Online]. Available: http://doi.acm.org/10.1145/2187980.2188222

[20] Y. Fan, W. Wei, Y. Gao, and W. Wu, "Introduction and analysis of simulators of mapreduce," in *Trustworthy Computing and Services*. Springer, 2014, pp. 345–350.

[21] B. Dufour, K. Driesen, L. Hendren, and C. Verbrugge, "Dynamic metrics for java," *SIGPLAN Not.*, vol. 38, no. 11, pp. 149–168, Oct. 2003. [Online]. Available: http://doi.acm.org/10.1145/949343.949320