Cluster Computing Journal manuscript No.
(will be inserted by the editor)

MOMTH: Multi-Objective Scheduling Algorithm of Many Tasks in Hadoop

Mihaela-Catalina Nita · Florin Pop · Cristiana Voicu · Ciprian Dobre · Fatos Xhafa

Received: date / Accepted: date

Abstract A real challenge sits in front of the business solutions these days, in the context of the big amount of data generated by complex software applications: efficiently using the given limited resources to accomplish specific operations and tasks. Depending on the type of application dealing with, when trying to deliver a certain service in a specific time and with a limited budget, a sequential application may be redesigned in a convenient way so that it will become scalable and able to run on multiple resources. In this context, Many Task Computing (MTC) model brings together loosely coupled applications, composed of many dependent/independent tasks, which will work together for a common result. When asking for a certain service, the most frequently constraints addressed by the user are deadline and budget. However, depending on the tasks nature used in MTC, other constraints may also occur: tasks may be data intensive or computing intensive, independent or dependent, uni-processor or multi-processor. In this context, we propose a multi-objective scheduling algorithm of many tasks in Hadoop for Big Data processing, named MOMTH. The algorithm evaluation was realized in Scheduling Load Simulator, integrated in Hadoop and easy to use. We compared the proposed algorithm with FIFO and Fair Schedulers and we obtained similar performance for our approach.

Mihaela-Catalina Nita
PhD Student, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: catalina.nita@hpc.pub.ro

Florin Pop - Corresponding Author
Associate Professor, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: florin.pop@cs.pub.ro

Cristiana Voicu
PhD Student, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: cristiana.voicu@hpc.pub.ro

Ciprian Dobre - Corresponding Author
Associate Professor, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: ciprian.dobre@cs.pub.ro

Fatos Xhafa, PhD in Computer Science Professor,
IUniversitat Politècnica de Catalunya
Barcelona, Spain,
E-mail: fatos@lsi.upc.edu

Keywords Task Scheduling · Hadoop · MapReduce · Many Task Computing · Big Data · Cloud Computing.

PACS Distribution theory, 02.50.Ng

Mathematics Subject Classification (2000) 68M20 · 68M14 · 68U20

1 Introduction

We are facing an explosion of data generation by almost everything that surrounds us: mobile devices, software applications, wireless sensors, electronic equipments and so on. A frequent situation of big data generation is met in the meteorological applications, were a simple mistake in data interpretation can lead to natural disasters. Normally one would think that these are the situations when dealing with big data. Actually, is not. A well-known coffee-shop has decided to improve it's image in front of the customers by using all the data obtained from simply selling the coffee in his own benefit. The data was analyzed and various solutions and models were proposed by researchers. In the end, the results were amazing: they managed to turn disloyal customers into loyal customers. What can we conclude from the two above examples is that there is value in every data and for every business case, not matter if we are talking about complex scientific applications or about simply restaurants. The challenge is: how we unlock it in an efficient way by keeping in mind all the constraints that a customer may raise? The two most common constraints are time and budget, which are always limited. However there are also other constraints like: data dependency or computational dependency. Not only once, the customer applications needed to deal with are complex, based on other smaller applications, dependent or independent, with specific business case constraints. In this case we need to find a solution for transforming all this sequential tasks in a such way that will decrease the execution time and it will minimize the number of resources used.

The goal of this paper is to design a multi-objective scheduling algorithm for many tasks, in Hadoop, named MOMTH (Multi-Objective scheduling algorithm of Many Tasks in Hadoop). In general, many complex problems are treated as sigle-objective optimization problem instead multi-objective problem by transforming all but one objectives into constraints. The novelty of current approach is that we will consider objective functions related to users and resources in the same time and constraints like deadlines (scheduling in due time) and budget. This paper presents an extended approach of MOMC algorithm [16]. The experimental results were driven in the Scheduling Load Simulator, already implemented in the Hadoop framework. The simulator takes the workload from the jobs history and shows in real time the obtained results based on different metrics.

Offering support for Big Data computation, Hadoop was chosen as the business bottom level for many software companies existing on the market. Since every company has it's own needs and requirements, each personalized and built their own scheduler. Although, the most popular methods are: FIFO scheduler, Fair scheduler and Capacity scheduler [18] [7]. It is important to mention that there are also other schedulers, with other objectives:scheduling to meet deadline [8], the total budget, data dependencies, computational dependency, etc. [19].

The killer application chosen to evaluate MOMTH algorithm is MobiWay, a collaboration platform that expose interoperability between a large number of sensing mobile devices and a wide-range of mobility applications, in order to provide useful information about the mobility like traffic conditions in a certain area or less crowded places. MobyWay is an obvious application based on the MTC Model: it bring together less coupled applications (taxi companies, delivery fleets) that will work together of a common purpose.

The rest of the paper is structured as follows. The second section describes the existing schedulers in Hadoop: FIFO, Fair Scheduler, Capacity Scheduler and HOD Scheduler. It also take a short overview of the existing solutions for multi-objective and many tasks computing scheduling. The following section introduces the proposed algorithm, MOMTH, by presenting the mathematics model behind it. MOMTH algorithm is based on two objectives: avoiding resource contention and having an optimal workload of the cluster, and two constraints: deadline and budget. The next section is dedicated to the experimental results and the used metrics, organized in two subsections: one that will focus more on the evaluation metrics and experimental setup and the other one that will presents the results, compared with FIFO and Fair algorithm. The final section will sum up the main ideas highlighted and it will propose further investigations.

2 Related Work

Many Tasks Computing (MTC) represents the connection between two novel computing paradigms: high throughput computing and high performance computing [12]. It differs from high throughput computing by using multiple computing resources for short periods of time, in order to accomplish a significant number of computational tasks. On the other hand, it differs also from the high computing paradigm by comprising multiple distinct and complex applications. MTC paradigm permits multiple applications to collaborate for a common cause. In this context, the task scheduling becomes a real challenge and it has been the source of inspiration of many researchers along time.

In [6] a particle swarm optimization (PSO) algorithm which is based on small position value rule is proposed, with the main target of decreasing the total processing time. It is based on the idea of optimizing the transferring and processing time, as being crucial for the total computational cost of an application.

Simon et al. propose in [13] a single queue algorithm based on greedy knapsack with dynamic job priorities. This algorithm is destinated to the HPC systems and it does not require user intervention in order to establish a certain job priority. The system performs an exhaustive parameter search in order to calculate a job priority. This approach fits best for applications that are based on multiple independent tasks.

In [9] a Hybrid Scheduler (HybS) algorithm based on dynamic priority is proposed. The target is to reduce the latency for variable length concurrent jobs, while maintaining data locality. The dynamic priorities are designed for different job sizes, task lengths or waiting times. It applies a fractional knapsack algorithm for the processor assignment. The evaluation shows that HybS improves the average response time for the workloads with an approximately 2.1 speed-up value.

Pandey et al. address in [10] the problem of single source data retrieval, which is not optimal in cases like: tasks are interdependent on data, the average size of data processed by most tasks is large and data transfer time exceeds task computation time. For the above issues they proposed to leverage the presence of replicated data sources and retrieve data in parallel from various locations. The number of locations used in the proposed model is two. The assignation of interdependent tasks is based on metrics like data retrieval time and computation time. The experiments were realized based on real applications in both real and emulated environment. The results showed a decrease in the total computational time.

At the beginning Hadoop had only one scheduler, but in time it proved to be inflexible to the business requirements. The solution found then was to implement a pluggable scheduler system. The solution was a real success since it offers more flexibility and it's also easy to integrate with Hadoop [17].

FIFO scheduler is the first Hadoop scheduler and it respect the FIFO model: a queue of tasks is implemented and the tasks are distributed to the masters by the arrival order. It represents the basic scheduling algorithm and it does not respect any constraints. *Fair scheduler*, developed by Facebook, gives each job equal resources, by helping small tasks to run in parallel with other tasks that requires more CPU. The main concern for this scheduler is resource congestion when dealing with a bigger number of tasks. *Capacity scheduler*, developed by Yahoo, is similar to the fair scheduler but with the major difference of using prioritized queue jobs. *HOD Scheduler* was designed to easy provision and manage Hadoop MapReduce and HDFS instances on a shared cluster of commodity hardware. To allocate a node, it uses Torque resource manager [14], and on each node two daemons are started: Hadoop MapReduce and HDFS.

YARN (Hadoop 2.0) splits in different components the Hadoop job tracker, the resource manager and the functionality used to schedule the jobs [15]. YARN comes with a novel approach where applications developed to be executed in a Hadoop environment contain the scheduling policy. Hadoop and YARN is used for various purposes which need high performance processing of data: Facebook tasks execution with Corona [3], human genome decoding, e-science simulations, etc.

Different multi-optimization problems of task scheduling in a Hadoop environment consider extension of MapReduce scheduling mechanism with the following aspects [5]: 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 used in inter-cluster communication, which are not considered in our approach, are presented in [20].

3 MOMTH Scheduling Algorithm

The MOMTH model is as follows. Each job J that will be executed in a Hadoop cluster has a known number of map tasks and reduce tasks, N_{map}, respectively N_{reduce}. The total number of tasks $N = N_{map} + N_{reduce}$ is known:

$$J = \{T_i | T_i \in Map \vee T_i \in Reduce\}, |J| = N. \tag{1}$$

Each job has an arrival time, A, a deadline, D and an allocated budget, B. For each job, we know from the beginning the exact amount of data that should be processed, In. The data processed by reducers is not known, but we can approximate it using the previous jobs. We call r the ratio that filter the amount of input data, $Out = r * In$. We consider that each job has an $Owner$, which handle the job and can give a rank, Π, about the execution. The properties of a job J may be expressed as:

$$Prop(J) = (A, D, B, In, r|Out = r * In, Owner) . \tag{2}$$

Each job J has a specific number of slots assigned, some of them for maps Sl_{map} and others for reducers Sl_{reduce}. We can use the same slot for a mapper and then for a reducer. In this case, the total number of slots is $Sl \leq Sl_{map} + Sl_{reduce}$.

Each task T_i can be associated with the following attributes:

– a processing time p_i, apriori evaluated for a map tasks or a reduce task; it is used to compute the total flow-time for a job:

$$Flowtime(J) = \sum_{T_i \in J} pi, \tag{3}$$

– a budget b_i, allocated by the owner for a job J. We define the task-specific budget b, and the normalized task-specific budget as:

$$b = \frac{B}{N}, \tilde{b}_i = \frac{b_i}{b}, \tag{4}$$

the performance of budget estimation being expressed as $\tilde{b}_i \leq 1, \forall T_i \in J$.

– a deadline d_i introduced for each task by the cluster resource manager to handle the synchronization between task execution:

$$\max_{T_i \in J} \{d_i\} \leq D. \tag{5}$$

– a weight w_i corresponding to the resources consumed that can be proportional with the product between processing time and budget allocated of a task:

$$w_i = \alpha p_i b_i = \alpha p_i \tilde{b}_i \frac{B}{N}, \tag{6}$$

where α is a proportionality constant. If $\alpha leq 1$ then the task schedule and execution will respect the total budget, otherwise the total cost may exceed the allocated budget. Considering the performance of budget estimation we have:

$$w_i \leq \alpha p_i \frac{B}{N}, \tag{7}$$

$$W(J) = \sum_{T_i \in J} w_i \leq \alpha \frac{B}{N} \sum_{T_i \in J} p_i = \alpha \frac{B}{N} Flowtime(J). \tag{8}$$

– a value π_i corresponding to the feedback of the owner. The owner can rank all tasks with the same value π. The global ranking vector is:

$$\Pi = [\pi_i]_{T_i \in J} . \tag{9}$$

We can compute the value π by measuring the performance of task execution. For example, we can consider the real execution time $cost_e(T_i)$ and the task completion C_i to estimate the ranking value:

$$\pi_i \propto \frac{p_i}{cost_e(T_i)} \times \frac{C_i}{d_i}. \tag{10}$$

These values and weights can be dynamically assigned after the task is generated based on the task characteristics and owner context. The properties of a task T_i may be expressed as:

$$Prop(T_i) = (p_i, b_i, d_i, w_i, \pi_i, type = map|reduce)\,. \tag{11}$$

The multi-objective optimization problem addressed by MOMTH is to satisfy multiple constraints and as much objectives as possible. The general problem is to find a vector x of N decision variables (equal with number of tasks), with $x_i^{(L)}$ and $x_i^{(U)}$ the lower and upper bound for each decision variable, that satisfies constraints (Q inequalities and K equalities) and optimize a vector of O functions whose elements represents the objective functions:

$$\begin{aligned} min \mid max: \quad & F(x) = [f_1(x) \quad f_2(x) \quad \dots \quad f_O(x)]\,, \\ subject\ to: \quad & G(x) = [g_1(x) \quad g_2(x) \quad \dots \quad g_Q(x)] \geq 0, \\ & H(x) = [h_1(x) \quad h_2(x) \quad \dots \quad h_K(x)] = 0, \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad \forall i = 1 \dots n. \end{aligned} \tag{12}$$

MOMTH algorithm consider the following multi-optimization problem:

$$\begin{aligned} min: \; & \sum_{T_i \in J} p_i x_i, \\ max: \; & \sum_{T_i \in J} \pi_i x_i, \\ subject\ to: \; & B - \sum_{T_i \in J} b_i x_i \geq 0 \Longrightarrow \frac{1}{N} \sum_{T_i \in J} \tilde{b}_i x_i \leq 1, \\ & D - max_{T_i \in J}\{C_i\} \geq 0, \\ & 0 \leq x_i \leq \tilde{x}_i, \quad \forall i = 1 \dots n. \end{aligned} \tag{13}$$

We consider in the MOMTH model that the objectives are more related to what the user wants to obtain. The *max* objective is related to user feedback. Moreover, the user specifies the values for deadline and budget. To implement the constraints mentioned above, the information about map and reduce tasks is needed, as well as information about the processing nodes. We need to know CPU, memory and IO transfer rate.

Since MOMTH is designed for a Hadoop environment, we did not consider the amount of data transferred from outside. We considered that the input is already placed in HDFS and are available at run time for any job. Only the processing cost and communication cost between mappers and reducers. We consider the most important objectives to satisfy: avoiding resource contention and having an optimal workload of the cluster. Regarding the constraints, these can vary, but since processing time and money are the most important factors, we finally considered deadline and budget. The previous version of the proposed algorithm, MOMC [16], considered that the decision variable can take only 0 or 1 value (0 when the resource is not selected for a task, 1 when we can decide the schedule). In the current form, the optimization problem considers the amount of a resource to be used. The decision variable has now a maximum value $\tilde{x}$, representing the weight of resource usage.

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 all jobs that need to run at a certain moment and the available resources.

Algorithm 1 MOMTH Scheduling Algorithm

```
Require: $R$ - the set of working nodes; $|R| \geq Sl_{map}^{(min)} + Sl_{reduce}^{(min)}$.
Require: $taskQueue$ - a queue where all tasks $T_i$ belonging to a job $J$ are inserted.
Require: $assignedTasks$ - a queue with assigned tasks for each worker node.
 1: for each $workernode \in R$ do
 2:     $maxService \leftarrow service(workernode)$;
 3:     $W \leftarrow 0$;
 4:     for each $assignedTask \in workernode.assignedTasks$ do
 5:         $newAssignedTasks \leftarrow workernode.assignedTasks - assignedTask$;
 6:         while $taskQueue$ is not empty do
 7:             $task \leftarrow taskQueue.deq()$;
 8:             $w_i \leftarrow weight(task)$;
 9:             $newAssignedTasks \leftarrow newAssignedTasks \bigcup task$;
10:             if $service(newAssignedTasks) > maxService$ then
11:                 $maxService \leftarrow service(newAssignedTasks)$;
12:                 $W \leftarrow W + w_i$;
13:             end if
14:         end while
15:     end for
16:     $workernode.assignedTasks \leftarrow newAssignedTasks$;
17: end for
```

The proposed algorithm follows the best usage jobs-resources strategy (see Algorithm 1) The *service* function should return a positive value if there are enough mappers and enough reducers in order to finish the job in the specified budget and until the deadline ends. 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 Sl_{map} and Sl_{reduce} can be provided by a specific cluster of resources. We define $cost(map)$ and $cost(reduce)$ as the time cost to process the amount of data by mappers, respectively by reducers. Also, we define $budget(map)$ and $budget(reduce)$, 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 (only for data existing in the cluster). So, we should define the time cost for data, $cost(data)$, and budget cost for data transferring and processing, $budget(data)$.

Let us consider $start(map)$ and $start(reduce)$, the start time for the mappers, respectively for the reducers. We can consider that $start(map) = A$ meaning that all mappers start when a job arrive in the internal queue. To compute the number of map slots needed by the job J, we need to know the maximum value of $start(reduce)$, $start^{(max)}(reduce)$:

$$Sl_{map}^{(min)} = \frac{cost(map)}{start^{(max)}(reduce) - start(map)}, \tag{14}$$

$$Sl_{reduce}^{(min)} = \frac{r * cost(reduce)}{A + D - r * cost(data) - start(reduce)}. \tag{15}$$

Based on the time processing cost, we can consider that the used budget is the multiplication between time cost and number of resources:

$$B_{used} = cost(map) * Sl_{map}^{(min)} * b_{map} + cost(reduce) * Sl_{reduce}^{(min)} * b_{reduce}. \tag{16}$$

The MOMTH algorithm can be integrated in Hadoop as follows. Since *JobTracker* is an independent component, the new scheduler should extend `TaskScheduler` class, along with the properties and the methods. When the implementation is done, the new 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 a new implementation, you just have to modify the configuration and specify it.

4 MobiWay Application for Hadoop

Urban mobility is an attendant of modern city life and its sustainability heavily depends on the consumption of different resources, such as electricity and fuel; therefore, it manifests in carbon dioxide (CO_2) and local pollutant emission and directly influences the quality of life of the citizens [11]. Urban traffic is responsible for 40% of CO_2 emissions and 70% of emissions of other pollutants arising from road transport [1]. Road transport is responsible for over 70% of the GHG emissions from the entire transport sector, and for 13% of total emissions of all sectors (transport sector represents almost 20% of total emissions) [1]. Intelligent Transportation Systems (ITS) aim to overcome such problems, through solutions that combine specialized infrastructure, communication technologies and innovative services, to enhance the quality of traffic and mobility management, multi-modal or road transport solutions.

However, in order to develop accurate traffic model, to be able to understand transportation phenomena and be able to make decisions from statistical models of how traffic flows tend to behave in real-world cities, you need a vast amount of data. Unfortunately, this exact amount of data required to construct accurate traffic models today acts as a barrier to prototype implementation for many ITS concepts, on city-level scales. In MobiWay, national funded project PN-II-PT-PCCA-2013-4-0321 No. 16/2014, we target the development of a robust, open-service and standards-based collaboration platform to expose interoperability between a large number of sensing mobile devices and a wide-range of mobility applications. Its goal is to provide a framework of ecosystems, which will allow to engage urban communities in exploiting the shared value of mobility that can be leveraged beyond the classical views of social networks, respectively the current trends of service creation. To facilitate collaboration, the framework aims to distribute different sources of information (coming from end users or ITS services, such as taxi companies, delivery fleets, etc.) within a service cloud composed from application providers. As various types of sensor data will be provided by mobility users, intelligent data gathering, processing and sharing have to be guaranteed between a mesh of hundreds of thousands of participants and the co-existing services with diverse interests. The project builds an ecosystem of integrated services and applications, flexible and reconfigurable, offered under multiple packages offered to users. The MobiWay's architecture is presented in Figure 1.

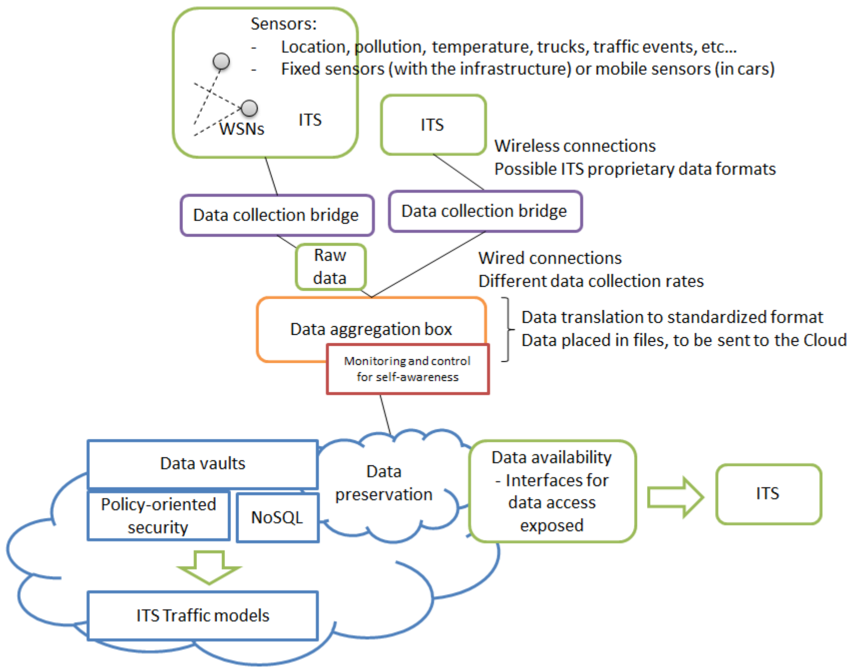
Sensors:
- Location, pollution, temperature, trucks, traffic events, etc...
- Fixed sensors (with the infrastructure) or mobile sensors (in cars)
WSNs
ITS
ITS
Wireless connections
Possible ITS proprietary data formats
Data collection bridge
Data collection bridge
Raw data
Wired connections
Different data collection rates
Data aggregation box
Monitoring and control for self-awareness
Data translation to standardized format
Data placed in files, to be sent to the Cloud
Data vaults
Policy-oriented security
NoSQL
Data preservation
Data availability - Interfaces for data access exposed
ITS
ITS Traffic models

Fig. 1 MobiWay concept and architecture

An important component in this architecture is Data Aggregation, which aims to process huge amounts of traffic data such that to deliver accurate Traffic data models at city levels. This means data coming from millions of cars in the city is collected, and persistently stored for longer time. This data is further, periodically, processed, resulting in statistical and probabilistic models of how likely is the traffic, depending on parameters such as time-of-the-day, weather conditions, scheduled events, and so on (simulated conditions in our case). Of course, such high-volume data processing raises problems, and we develop currently the algorithms to deal pre-processes such as data aggregation, cleansing, statistical analysis at different granularity levels (analyse the macro-level traffic in different areas of the city, which is next followed by micro-level, fine-grained, evaluation of traffic conditions – at street level).

The processing application uses the MapReduce paradigm, and Apache Hadoop and Spark as supporting technologies. 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 MobiWay, we thus needed to implement both map and reduce functions using the appropriate interfaces that would deliver optimized performance (we want to obtained a solution both cost and time optimum).

5 Experimental Scenarios

This section presents comparative results of MOMTH algorithm and all the setup steps needed to develop and then to test the new 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.

The Hadoop cluster setup considers the following configuration aspects:

- Hadoop cluster, developed on a machines with Ubuntu, a 64 bits architecture with maven, libssl-dev, cmake and protobuf libraries;
- HDFS configuration, with `dfs.replication` set to support fault tolerance;
- Scheduling Load Simulator (SLS) setup (the entire simulator memory or the one for each container, the number of virtual cores, etc.);
- 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 SLS.

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 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.

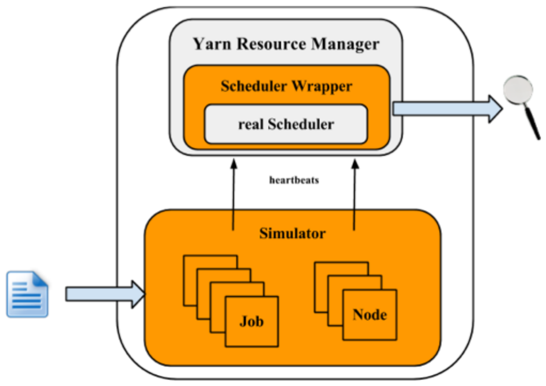

Fig. 2 Scheduling Load Simulator Architecture

Figure 2 shows the architecture of SLS. As we previously mentioned, everything runs on a single machine. This means that all Hadoop 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.

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.

There are two steps in the transformation process: `TraceBuilder` 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 [4].

5.1 Performance Metrics

The scheduling load simulator use Metrics library [2] to evaluate the results of a scheduling algorithm. We present the metrics used by SLS to show the results of MOMTH algorithm compared to most used algorithms, Fair and FIFO. 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.

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 5000 tasks (MobiWay), which describe the usage of proposed applications on many devices (generating many tasks) during a specific period of time (2 hours).

5.2 Experimental Results

We run 5000 tasks 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 12 nodes (quad core, dual processor).

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

Figure 3 presents the number of allocated cores for a node in the cluster. The core usage is very similar for Fair and MOMTH schedulers. The results correlated with job execution time (Figure 11) show that Fair and MOMTH has the best cores allocation scheme.

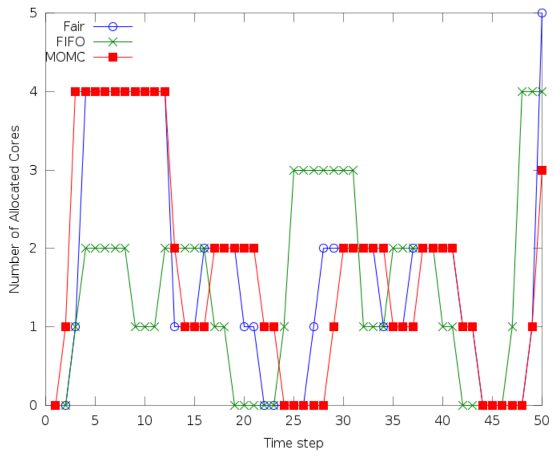

Fig. 3 Number of allocated cores for a node in Hadoop Cluster

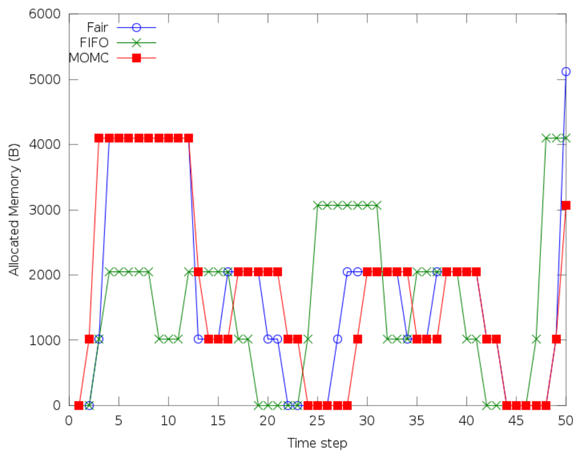

Fig. 4 Allocated memory for a node in Hadoop Cluster

When a job uses more memory for map or reduce task, the simulator memory will increase. We can clearly see (Figure 4) that FIFO and FAIR used more memory because they run the job with extra big input. This is another aspect which shows that MOMTH do not run the tasks which exceeds the deadline, the budget or both of them.

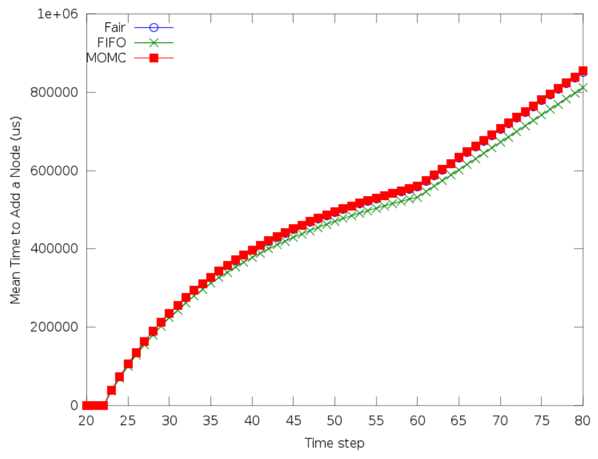

Fig. 5 Performance of resource provisioning in Hadoop Cluster

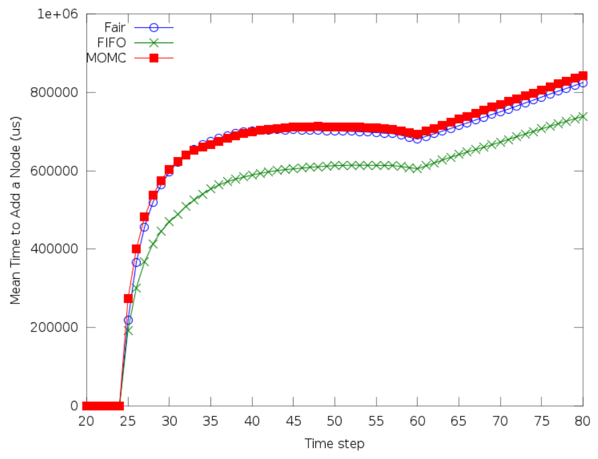

Fig. 6 Performance of resource reconfigure/update in Hadoop Cluster

Figures 5 and 6 presents the performance of resource provisioning and resource reconfigure/update in Hadoop Cluster. We can observe that the cost time for MOMTH approach is similar with Fair and FIFO, but it is normal to be higher than the others two because MOMTH compute dynamically the number of mapper and reducer slots.

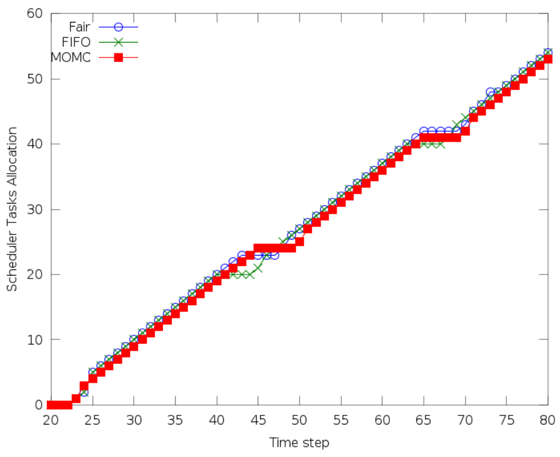

Fig. 7 Tasks allocation scheme

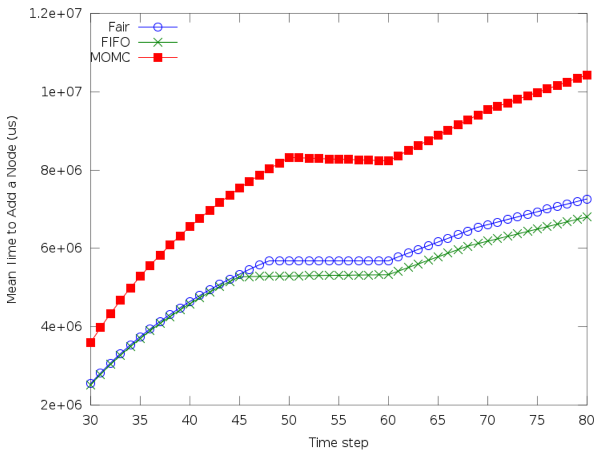

Fig. 8 Cost for application setup in Hadoop Cluster

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, the jobs are executed in the same ordered as they are submitted. In case of, the scheduler rearranges the jobs because all of them should have access to the resource, even if one has more input. Regarding MOMTH 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.

Figure 7 shows that all tested scheduling algorithms allocate in the same meaner the tasks in the cluster, so there are no waiting tasks staying in a queue for a long period of time.

These metrics present the time cost for each operation: allocate, node added, node removed, node update, application added, application removed, container expired.

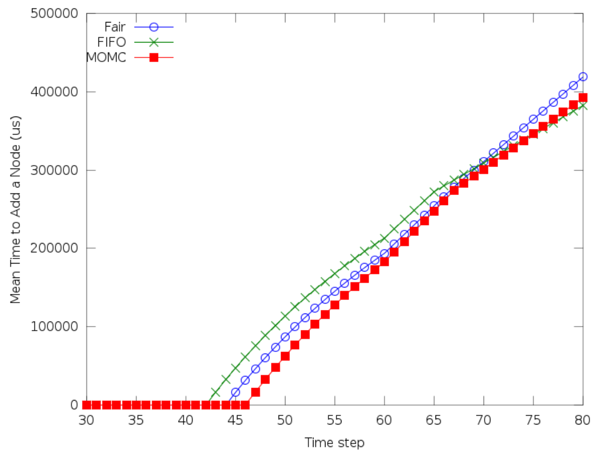

Fig. 9 Cost for application finish in Hadoop Cluster

As you can see in figures, during the tests only three operations were done: node added and node updates, application added and application removed. During the entire execution time, the cost for these operations is lower in MOMTH case.

Figures 8 and 9 presents the performance of task setup in the Hadoop Cluster. It is clear that MOMTH obtained the highest time but this is explained by the complexity of the algorithm in steps 6-12 in Algorithm 1.

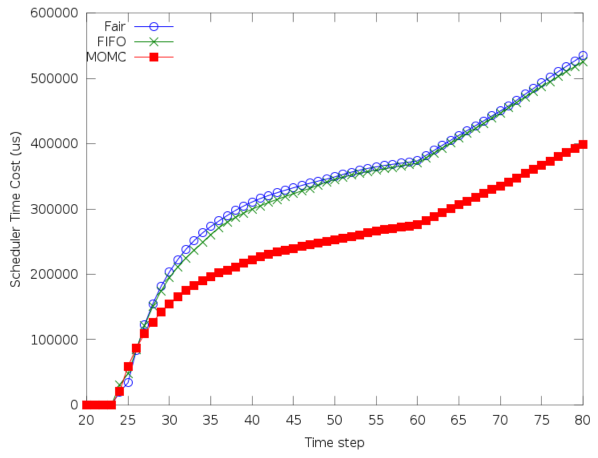

Fig. 10 Time cost for Scheduling Phase

The scheduler time cost is presented in Figure 10. The MOMTH obtained the better performance due to the optimization criteria considered. The results are similar for FIFO and Fair schedulers because the approach of tasks scheduling is similar. Correlated with results for application added, Figure 11 presents the start time,

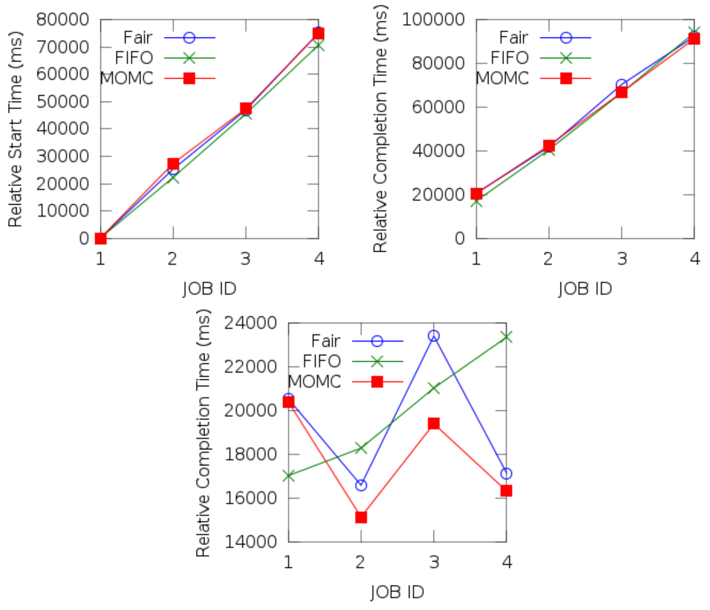

Fig. 11 Runtime for all scheduled tasks: start time, completion time and duration

completion time and duration for all scheduled tasks. The relative times are similar which means that we can obtain the same performance with multi-optimization algorithm like a classical one.

6 Conclusions

This paper proposed a new algorithm named MOMTH, 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 MOMTH 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 MOMTH 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 MOMTH 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. István László Bart. Urban sprawl and climate change: A statistical exploration of cause and effect, with policy options for the {EU}. *Land Use Policy*, 27(2):283 – 292, 2010. Forest transitions Wind power planning, landscapes and publics.
2. Bruno Dufour, Karel Driesen, Laurie Hendren, and Clark Verbrugge. Dynamic metrics for java. *SIGPLAN Not.*, 38(11):149–168, October 2003.
3. Facebook. Under the hood: Scheduling mapreduce jobs more efficiently with corona. http://goo.gl/XW9nD7, October 2012.
4. Yuanquan Fan, Wei Wei, Yan Gao, and Weiguo Wu. Introduction and analysis of simulators of mapreduce. In *Trustworthy Computing and Services*, pages 345–350. Springer, 2014.
5. Rong Gu, Xiaoliang Yang, Jinshuang Yan, Yuanhao Sun, Bing Wang, Chunfeng Yuan, and Yihua Huang. Shadoop: Improving mapreduce performance by optimizing job execution mechanism in hadoop clusters. *J. Parallel Distrib. Comput.*, 74(3):2166–2179, March 2014.
6. Lizheng Guo, Shuguang Zhao, Shigen Shen, and Changyuan Jiang. Task scheduling optimization in cloud computing based on heuristic algorithm. *Journal of Networks*, 7(3):547–553, 2012.
7. Zhenhua Guo and Geoffrey 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)*, CCGRID '12, pages 714–716, Washington, DC, USA, 2012. IEEE Computer Society.
8. Kamal Kc and Kemafor Anyanwu. Scheduling hadoop jobs to meet deadlines. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 388–392, Washington, DC, USA, 2010. IEEE Computer Society.
9. Phuong Nguyen, Tyler Simon, Milton Halem, David Chapman, and Quang Le. A hybrid scheduling algorithm for data intensive workloads in a mapreduce environment. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, UCC '12, pages 161–167, Washington, DC, USA, 2012. IEEE Computer Society.
10. Suraj Pandey and Rajkumar Buyya. Scheduling workflow applications based on multi-source parallel data retrieval in distributed computing networks. *Comput. J.*, 55(11):1288–1308, November 2012.
11. EU Parliament. Resolution of 10 september 2013 on promoting a european transport-technology strategy for europe's future sustainable mobility. http://bit.ly/1vJm2Ho, October 2014.

12. I. Raicu, I.T. Foster, and Yong Zhao. Many-task computing for grids and supercomputers. In *Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on*, pages 1–11, Nov 2008.
13. Tyler A. Simon, Phuong Nguyen, and Milton Halem. Multiple objective scheduling of hpc workloads through dynamic prioritization. In *Proceedings of the High Performance Computing Symposium*, HPC '13, pages 13:1–13:8, San Diego, CA, USA, 2013. Society for Computer Simulation International.
14. Garrick Staples. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.
15. Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM.
16. Cristiana Voicu, Florin Pop, Ciprian Dobre, and Fatos Xhafa. Momc: Multi-objective and multi-constrained scheduling algorithm of many tasks in hadoo. In *3PGCIC-2014, The 9-th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing.* IEEE Explore, November 2014.
17. Yang Xia, Lei WANG, Qiang ZHAO, and Gongxuan ZHANG. Research on job scheduling algorithm in hadoop. *Journal of Computational Information Systems*, 7(16):5769–5775, 2011.
18. Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.
19. Fan Zhang, Junwei Cao, Keqin Li, Samee U. Khan, and Kai Hwang. Multi-objective scheduling of many tasks in cloud platforms. *Future Generation Computer Systems*, 37(0):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.
20. 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*, 80(5):994–1007, 2014. cited By (since 1996)0.