MLBox: Machine Learning Box for Asymptotic Scheduling

Mihaela-Andreea VASILE[a], Florin POP[1a], Mihaela-Cătălina NIŢĂ[a], Valentin CRISTEA[a]

[a]*Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Romania*
Emails: mihaela.a.vasile@gmail.com, florin.pop@cs.pub.ro,
catalina.nita23@gmail.com, valentin.cristea@cs.pub.ro

Abstract

As the usability of Cloud-based solutions has increased for various types of users with different needs, from scientists that want to process big data sets collected from sensors or business analysts that want to take decisions based on the huge amount of gathered data to simple users that store or share documents via a Cloud platform, the generated data is increasing more and more. For example, the ATLAS and other detectors at CERN generate petabytes of data and Facebook stores data with a rate of around 600 TB daily. In the current context, efficient scheduling for Big Data applications is a challenge and an appropriate scheduling technique is required for different types of incoming requests. In this paper we propose a scheduling algorithm for different types of computation requests: independent tasks, like bag of tasks (BoT) model or tasks with dependencies modeled as directed acyclic graphs (DAG), and they will be scheduled for execution in a Cloud datacenter. The tasks in the requests are scheduled on the available resources using the suitable scheduling algorithm for each request. We rely on a machine learning toolbox, named as MLBox, to find what algorithm should be used for a certain request. We implemented four heuristics for scheduling BoTs and four heuristics for DAGs scheduling and generated the training data for the machine learning algorithm by running multiple traditional scheduling algorithms and selecting the 'best' one for a given request. We evaluate the performance by comparing the scheduling of different tasks requests using some of the traditional algorithms and our machine learning based scheduling algorithm.

Keywords: Asymptotic Scheduling, BoT Scheduling, DAG Scheduling, Scheduling Heuristics, Machine Learning, Datacenters

1. Introduction

The scheduling problem refers to assigning tasks on resources in an almost optimal manner. This is one of the key topics in the current context of distributed systems as the schedulers, part of Cloud platforms (like Apollo [4], Bistro [14], Paragon [10], DejaVu [38], etc.), have to face new challenges caused specially by Big Data applications like processing huge amounts of heterogeneous data, generated in a short period of time, via data flows or on-line streaming, which has to be managed very fast (read, write, filter, compute statistics).

Cloud solutions enable users the access via Internet to various types of resources such as existing applications in the Cloud, frameworks that can be used for development of custom built

[1]Corresponding author, Email: florin.pop@cs.pub.ro

applications (Hadoop, Spark, Flink, RabbitMQ, etc.), access to Virtual Machines (VMs) for installing operating systems and also storage and sharing solutions. Therefore, the Cloud is now a significant choice for multiple types of users, be them common individuals, scientists or technical users. The Big Data applications handle huge amount of data, and scheduling such applications is a real challenge [12, 27]. For example, Hadoop queries for data retrieval in map-reduce applications may generate a significant number of tasks because they are generally used on very large data sets.

In the homogeneous distributed systems the resources have similar properties and tasks execution on different processing units will take the same time. The heterogeneous systems consist of resources with different characteristics so the execution time of one task on different machines will vary. Usually Cloud datacenters are homogeneous because the resources administration is easier if they are identical and the security issues are mainly related to the overall system architecture [45] so the traditional Cloud systems contain identical commodity machines. Nevertheless, heterogeneous Cloud systems have been developed [7, 36], because this model allows to shift from one architecture to another as needed providing the following advantages: it is possible process more data faster (change the architecture and approach for different types of requests), and provide access to different types of groups (smaller research groups will also afford the access to Cloud systems) [28, 26, 34].

The motivation for approaching the scheduling problem in this paper is the increasing need of efficient processing of very large data sets [18, 42]. Examples of applications that generate such large data sets include the Montage general engine [2] that computes the mosaic of input images. The input images are generated by astronomical projects such as IPHAS [11] (INT Photometric Hα Survey of the Northern Galactic Plane) that displays the northern part of the Milky Way in visible light, which database contains more than 2 million images with a size of around 2 TB. In [5] it is described the detailed costs of producing mosaic of the input images, which raises to thousands of $. In this case, even small optimizations will impact the overall costs. Other applications could generate data of petabytes scale, like the ATLAS and other detectors at CERN [5, 9].

The area of genotype research generates very large data sets. For example, TCGA (The Cancer Genome Atlas) stores the normal and affected tissue for hundreds of for over 20 different cancer types. In this case, the genomic data size is of about 500 TB and is expected to grow to PB scale in the next 3 years. Therefore we have a growing need for environments that can manage the analysis of these large amounts of data sets, and one solution is be to have cloud-based access to them. Such a Cloud solution, part of the Open Science Data Cloud, is Bionimbus Protected Data Cloud (BPDC) [19].

Between the generators of Big Data sets we can also find a large number of non-scientific and non-technical applications such as: Facebook, LinkedIn or Instagram. For these applications, a huge number of users execute different types of actions concurrently. For the Facebook example, users may upload photos, share pages or post comments. All these actions generate very large amounts of data, up to hundreds of PB, with daily incoming rates of TB stored as Hive data, and using Map Reduce for queries [37].

In this paper we propose a scheduling algorithm that addresses the following model: the requests may contain independent tasks or tasks with dependencies (or worflows) and they will be scheduled for execution in a datacenter. The developed scheduling algorithm uses a machine learning box (MLBox) tool that receives as input a request (containing a large number of tasks), the output is which scheduling algorithm should be used for this request (labeling techniques). Then, the tasks in the request is scheduled using the selected algorithm. The training data for

the learning algorithm consists in a very large set of labeled requests (billions), each containing thousands tasks. We built the training data as follows: for each request in the training set we run multiple traditional scheduling algorithms (for independent tasks or tasks with dependencies), select the 'best' algorithm (which optimizes a chosen cost function) and label the request with the selected algorithm. We validate the proposed algorithm by comparing the scheduling of different requests using some of the traditional algorithms and our machine learning based algorithm.

Our contributions in this paper are summarized as follows:

- We performed a critical analysis of existing traditional scheduling algorithms both for independent tasks and for workflows, for which types of tasks they are better suited. Based on this analysis we chose the algorithms for scheduling BoTs and DAGs that we will integrate in our new proposed scheduling algorithm.
- We implemented a machine learning algorithm used for labeling requests: select the best scheduling algorithm that will be used for schedule the tasks on the given resources. We also generated the training data for the machine learning box. We implemented the previous enumerated traditional algorithms, and chose one for each request.
- We proposed a novel hybrid approach for tasks scheduling in homogeneous Cloud systems based on labeling requests (sets of tasks) using the MLBox tool. This approach was used because we admit that there is no solutions that could fit all types of tasks models. Therefore, our algorithm is based on using different scheduling strategies, selected by considering the heterogeneity application tasks and/or flows.
- We extended CloudSim simulator [6] to integrate scheduling strategies for BoTs and DAGs, and also our machine learning based algorithm encapsulated in the MLBox tool.

The rest of the paper is structures as follows. In Section 2 we analyze the background and current state of the art: a short overview of the current applications that generates many tasks (with and extended 8-V model for Big Data) and a critical analysis of current scheduling algorithms for different types of requests, presenting advantages and possible improvements. A detailed description of our solution is provided in Section 3: traditional scheduling algorithms that are used, the proposed machine learning based scheduling algorithm, and the MLBox tools that we developed. In Section 4, we present the implementation details, which include the integration with the CloudSim simulator. The experimental methodology, results and discussion of the results are covered in Section 5. Finally, in Section 6 we present the conclusions and also possible directions for future research.

2. Background and Related Work

Asymptotic task scheduling represents either the scheduling of continuous large workloads or workflows, as soon as they arrive at the system or the analysis of tasks scheduling for an "infinite" time horizon. Asymptotic scheduling analysis is related to Many Tasks Computing (MTC). The notion of MTC has been introduced by Ioan Raicu [30]. It is defined as a combination of High Performance Computing (HPC) and High Throughput Computing (HTC) and it refers to a very large number of various sets of tasks: independent workloads/workflows, small/large tasks, computational/data intensive (I/O rates, FLOPS, no tasks, dependencies). The sets of tasks arrive with great speed. The MTC notion includes multiple problem types, classified using the number

of tasks and the data sets size: Big Data (very large data sets and very high number of tasks), Map Reduce (very large data sets and relatively reduced number of tasks) and HTC (smaller data sets and very high number of tasks). So, Big Data may be considered as part of MTC, using this classification.

Asymptotic scale requests are encountered in Big Data platforms and they involve queries/data analytics on Big Data sets. The challenges are that these datasets cannot not be managed, processed or stored by traditional hardware/software solutions within a tolerable time due to the very large dimensions of the set, they are produced at a high speed and cannot be entirely stored. Big Data is often described using a multiple V's model [1]; the extended model of these V features is as follow (the 8-V model):

- *Volume* - Big Data usually refers TB up to PB scale data sets, that have to be acquired, stored and analyzed at different time intervals (daily, weekly) depending on the source.
- *Velocity* - the data may need to be acquired and processed/stored at different speeds. Some chunks may arrive as batch at certain time intervals, in a near-time manner (at small time intervals), real-time (continuous data) or as streams of data.
- *Variety* - Big Data sets structure can be defined by different data types: structured (well-defined data model), unstructured (data model is not defined), semi-structured (not strict data model) or mixed (various types).
- *Variability* - some Big Data sets face with constantly and repetitive changing the can have an important impact on data homogenization.
- *Veracity* - the data being stored should be 'cleaned', any noise or abnormality in the data should be removed such that the analysis performed on the data can be relied on.
- *Value* - Big Data can be used for multiple goals: reporting of business processes or transactions, churn analysis (why does user engagement drop), diagnosing system failures and also make different decisions.
- *Volatility* - this characteristic refers to how long should the data be stored, and is influenced by how long the data is valid and can be relevant for the current use.
- *Visualization* - refers to intelligent methods that transform spreadsheets and reports into dynamic charts and graphs used to visualize large amounts of complex data sets.

When Big Data applications are seen as read/withe operations to memories or storage devices, they create heavy workloads (BoTs or DAGs), being the best killer applications for asymptotic scheduling.

The scheduling problem is one of the key topics in the context of distributed systems, and it could be described as follows: consider a set of tasks, each task is composed by a number of operations, and a set of resources, a resource may be a physical/virtual machine or a processor, then we may define scheduling as the allocation of each task on a resource. A scheduling algorithm tries to optimize different measures, such as total execution time (minimize), CPU load (maximize) or optimize more complex cost functions. The continuous improvement in scheduling techniques is still and open issue. As the costs for storing and processing data sets is very high in the context of Big Data scales, every improvement on system performance has a significant

impact. More than this, with the efficient scheduling also means a better usage of the resources and reduced costs for consumed energy or data centers administration [32].

As the storage and computation systems evolve, the scheduling topic has approached and consider new techniques for the new challenges. The challenges are: variety of tasks and data types, data volume to be managed or the response time required. A class of independent tasks are BoTs requests, that contain a very large number of tasks that do not need to communicate during execution and their execution does not depend on the results of other tasks. Examples of applications that generate BoT jobs include intensive search applications - brute force for password breaking, Monte Carlo simulations or image processing. The tasks with dependencies impose additional restrictions in the scheduling process. The workflows describe a process (business or scientific) and a set of rules (dependencies) are also tasks with dependencies and may be described using a DAG with nodes for each task and edges for each dependency. Examples of scientific workflows are generated by applications including the Montage general engine [35] for computing the mosaic of input astronomical images or the Neptune project, that acquires measurements from ocean temperature/currents sensors and models them.

2.1. Independent Tasks Scheduling

The existing heuristic for independent tasks scheduling algorithms have some advantages considering that large amounts of tasks have to be processed: the speed is very good due to the fact that they do not search the best scheduling, but a good approximation, so it's suitable for large problems and the computed schedule is not the optimal, but a near-optimal one. In [21] six heuristic algorithms are described and compared: five of them are popular pure heuristic scheduling algorithms and a new heuristic is proposed. They are compared using the total makespan and flowtime.

Min-Min algorithm uses the minimum completion time as selection metric. The main idea is that the tasks that have the earliest end time, have the highest priority. For each task, the minimum completion time is computed across all machines and just one is selected. Next, the global minimum across all tasks is selected along with the corresponding machine. The task is removed from the list, assigned to the corresponding machine and the workload is updated.

Max-Min algorithm also uses the minimum completion time as selection metric, and is similar to the Min-Min algorithm. Here, the idea is that for each task, the minimum completion time is computed across all machines and just one is selected, and next, the global maximum across all tasks is selected along with the corresponding machine.

Both min-min and max-min algorithm have some limitations for certain types of job sets. When using min-min, the efficiency decreases when we have a larger number of 'small' tasks and for max-min when there are more 'large' tasks to be scheduled. The Longest Job to Faster Resource - Shortest Job to Faster Resource (**LJFR-SJFR**), comes as a compromise between the previous algorithms. For a number of m initial steps, the longest job (maximum completion time) is assigned to the faster resource, and next it alternatively assigns both shortest job (minimum completion time) and longest job to the faster resource.

The **Suffrage** algorithm computes the minimum and second minimum completion time found for each task and the suffrage value, defined as the difference between the two computed values. The global maximum suffrage value across all tasks is computed, and the task with largest suffrage value is assigned to the machine with minimum completion time for the selected task.

In the **Workqueue** algorithm, at each step, a random task is selected and removed from the list, it is assigned to the machine with the minimum workload and the workload is updated.

The **Min-Max** algorithm is the heuristic proposed in [21]. It uses a combined metric for scheduling evaluation based on the minimum completion time, which is computed for each task and also the minimum execution time. The evaluation metric for each task is the execution time on the machine with minimum completion time divided by the minimum execution time. The maximum ratio is computed across all tasks, and the selected task is assigned to the corresponding machine.

An agent-based Cloud scheduling approach for BoT is described in [16]. The agents involved in the scheduling scenarios are: Consumer Agents (CA), Resource Agents (RA), Broker Agents (BA) and Service Providers Agents (SPAs). The CA submit requests to multiple BAs, which send back a proposal. Then the CA selects the cheapest BA. In the next stage, the BA selects resources from multiple Cloud SPAs, and schedules the tasks with one of the described 14 heuristics. The communication between the brokers, service providers and resource agents is enabled via the contract net protocol (CNP). In the experiments presented in the paper, the approach handled heavy workloads if enough Cloud resources were available.

Another algorithm for scheduling independent tasks is HySARC^2 [39]. Its goal is to improve the workload on the available resources by taking into account the heterogeneity of the resources on the one hand (the resources are clustered and labeled) and the different types of tasks (the incoming tasks are also clustered). Therefore, the topics of interest for this paper are clustering, resource provisioning and hybrid scheduling that uses scheduling algorithms for different clusters of tasks.

2.2. DAG Scheduling

The DAG representation of a job with multiple tasks is $\mathcal{G}(V, E)$ where V is the set of tasks with a computation cost assigned (the nodes and weights) and E is a set of communication messages and dependencies which have assigned a communication cost (directed edges and weights) [22]. Most of the DAG traditional scheduling algorithms use the following attributes of a DAG to assign the tasks to resources: *t-level* of a node is the length of a longest path from an entry node reaching the current node, summing the node and edge weights from the path, called also static level (*sl*); *b-level* of a node is the length of a longest path to an exit node from the current node, summing the node and edge weights from the path; *ALAP* (As-Late-As-Possible) the maximum delay for the execution start time of node, so that the total schedule length is not delayed, called also Latest Start Time (LST); *ASAP* (As-Soon-As-Possible) is another name for the *t-level*, called also Earliest Start Time (EST).

In [17] four algorithms for DAGs are compared: two static algorithms and two dynamic ones. These algorithms are based on list scheduling: nodes are added to the list using some of the attributes above or a combination and the resource is selected to optimize a cost function. The static algorithms prioritize the tasks, and then select the resource and for the dynamic ones, these phases are not disjoint, so the resources are involved in task selection.

The Highest Level First with Estimated Times (**HLFET**) algorithm uses the static level attribute (*sl*). The first nodes to be scheduled are the entry nodes, which are sorted descending using this value. Until all tasks are scheduled, the first node in the list is scheduled, all the other nodes that became available are inserted in the list and the list is sorted again.

In the case of Modified Critical Path (**MCP**) the priority criteria is the ALAP of a node as the scheduling priority. The nodes are added to a list in ascending order of ALAP values and use the ALAP values of the children to break ties. Each task is then scheduled on the resource that allows earliest start-time using.

At each step, Earliest Time First (**ETF**) algorithm computes the global minimum execution start time across all ready tasks on each resource, selects the pair (node, resource), schedules the task on the corresponding machine and updates the list with any new 'ready nodes'. The algorithm starts with the entry nodes list.

A composite attribute for tasks priority is used in the Dynamic Level Scheduling (**DLS**) algorithm. The Dynamic Level (DL) is the computed as the difference between the *sl* and the EST, for each task on each resource. At each iteration, the DL is computed for all ready nodes and the pair with the largest DL value is selected. The difference between DLS and ETF is that ETF always schedules the tasks in ascending order of the EST, when DLS uses for start using the descending order of the sl and towards the end uses the ascending order of EST.

The Mobility Directed (**MD**) algorithm [41] uses a more complex metric in the scheduling process, named the relative mobility of a node that depends on the current Critical Path (CP) length, the *t-level*, *b-level* and the weight of the node. For nodes on the critical path this attribute has value 0. The algorithm starts with all nodes in a list, selects a node with minimum value for the relative mobility and has no predecessors with the same minimum value. After the node is scheduled on a machine using a condition for schedule a task on resources, the list of scheduled tasks for that machine can be updated, and also the DAG will be modified, by changing edges weights to 0 or adding new edges. So, at each step, the relative mobility value is recomputed for each node left to be scheduled and for the new DAG. A main advantage compared to the four previous algorithms is that an optimal schedule is generated even for DAGs with a large number of joins and forks (a large number of levels).

The Dynamic Critical Path (**DCP**) was proposed in [23]. It also has the advantage of generating an optimal schedule is for DAGs with large number of joins and forks. The algorithm is based on the node's mobility, a metric similar to the relative mobility, and the difference is that it does not depend on the node weight. DCP selects node with minimum difference between Absolute-Latest-Start-Time (ALST) and Absolute-Earliest-Start-Time (AEST) (same value as the node mobility). For resource selection it uses look-ahead strategy.

In [40], the authors propose a clustering based scheduling algorithm suitable both for independent and DAG tasks. The resources are profiled as they are added or removed, and the tasks are labeled as they arrive. The tasks are scheduled on clusters of machines according to the labeling. In this paper, the considered algorithms for DAGs are MCP and ETF.

In [29], a DAG scheduling algorithm based on genetic algorithms is presented. The chromosome representation is compact and easy to implement when facing crossover or mutation. Another characteristic is that the mutation rate of new chromosomes is dynamic and depends on the fitness functions variation. The fitness function in this paper is the balancing of workload for the resources.

The multiple priority queue genetic algorithm (**MPQGA**) proposed in [43], is a hybrid approach for scheduling DAGs that combines evolutionary (genetic) and heuristic algorithms. The genetic algorithm (GA) is used to assign priorities to the tasks. For mapping a task to a processor the ETF heuristic algorithm is applied. The MPQGA algorithm is evaluated using both task graphs from real-world scenarios, like Fast Fourier Transformation or Molecular dynamics and random DAGs.

In [25] a scheduling algorithm for workflows is proposed. The algorithm is based on a particle swarm optimization (PSO) heuristic applied for scheduling: Variable Neighborhood Search (VNS). In the paper, the theoretical model for workflows and PSO is described, and the scheduling context is mapped to the particle space (encoding step). The last step is to decode the optimization result to an actual schedule: the decoding step.

The Reliability Maximization with Energy Constraint (**RMEC**) algorithm [44] has been designed to address the scheduling problem in a heterogeneous system, considering possible failures of the applications, and trying to minimize the energy consumption. Tradeoffs are made between having a reliable system and a low energy consumption. The first important phase is to set priorities for tasks using two DAG computed properties: URank and DRank. Then, the processor is selected using the best frequency-voltage pair (lowest energy & highest availability). The final step is processor assignment. The evaluation tests were performed on random DAGs and real-world examples (FFT or LU DAGs).

In the domain of parallel and distributed systems in general, and scheduling in particular, machine learning techniques are used for enhancing the traditional algorithms with forecast elements [24, 8]. The elements that are subject to forecast include: system load, consumed energy, workflow forecast or resources status (churn analysis on machines) so the scheduling is more efficient.

In [20] the scheduling algorithm is based on a neural network (**NARX**) used for the prediction of the system load. The inputs for the method are the load intervals for the CPU and variance of future system characteristics (new resources may be added) and the output is the load decision for a machine (1/0). The output is used in the scheduling process.

The scheduling algorithm proposed in [15] intends to provide an efficient scheduling for high-loaded systems, that execute workflows. The algorithm detects patterns in recurrent tasks, and aims to provide as forecast the future workflow of the system, and uses multiple forecasting methods (SSA, ARIMA).

2.3. MTC Scheduling

Data intensive subset of MTC has been payed a lot of attention. In [31] has been proposed an approach for enabling data intensive many tasks computing, including: efficient acquiring of storage and computing resources in a dynamical way as the requests arrive, replicates the data depending on the demand and also schedules the computation on resources close to the data storage units. So, one of 'data diffusion' main contributions in the paper is the data-aware scheduler which implements four policies for dispatching tasks and three of them consider the data locality. In [33], the authors a hybrid algorithm for scheduling many-tasks computing. They designed an Asymptotically Optimal (**AO**) scheduling algorithm that depends on the existing scheduler in the system - they considered the case of First Come First Served (**FCFS**). Next, using a mathematical model, they determine the point when to switch from the default scheduler to AO as the job rate increases, and also the point when to switch back when the job rate decreases.

3. Defining the Box: Proposed Scheduling Algorithm and MLBox Tool Architecture

In this section we describe the model for tasks, resources and the proposed scheduling algorithm along with the used scheduling algorithms. Then, the architecture of MLBox and their components are presented. The necessary and sufficient conditions for a feasible scheduling solution are detailed.

3.1. Theoretical Model

A *Task* is defined as a set of operations that have to be executed on the same processing unit. We use four of the most common parameters to describe it formally:

$$T = (P_1^T, P_2^T, P_3^T, P_4^T), \tag{1}$$

where:

- P_1^T is the required CPU processing time by to execute the job on a resource;
- P_2^T is the IO time in which the task has to read the input in order to start the execution, and to write the output data when the execution is done;
- P_3^T is the preemption flag (preemptive or non-preemptive);
- P_4^T is the deadline.

For our model, we also have to define the *Requests* or *Jobs*, representing a set of *Tasks*, as follow:

$$\mathcal{T} = \left\{T_i, 1 \leq i \leq n | T_i = (P_1^{T_i}, P_2^{T_i}, P_3^{T_i}, P_4^{T_i})\right\}. \tag{2}$$

The set can contain independent tasks (BoT), or tasks with dependencies/restrictions (workflows). The BoT types of requests we need no additional definitions, as they are classified using the properties of the elements in the set. The requests that contain tasks with dependencies will be modelled as DAGs (Directed Acyclic Graphs). The formal definition of a DAG used in the paper is:

$$\mathcal{G} = \mathcal{G}(V, E, w, c), \tag{3}$$

where:

- V is a set of nodes representing the abstraction of tasks, so node n_i refers to task T_i. Each node is associated a weight (w), where $w(n_i)$ is the computational cost of task T_i (same as $P_1^{T_i}$);
- E is a set of directed edges that model the dependencies between tasks, noted as $e(n_i, n_j)$. A weight is also associated to each edge (we will refer to it as the cost of the edge), and $c(n_i, n_j)$ represents the time needed to transfer output data from task T_i as input data for task T_j. If the two tasks are executed on the same resource, the communication cost will be 0.

For the tasks that are part of a DAG request, we defined in the previous section multiple properties used in the scheduling algorithms. These properties depend on the DAG structure, the position of the node in the DAG and the weight functions. We identified simple properties, like the *b-level*, *t-level*, ASAP, ALAP or *sl* or different combinations of these metrics, like the mobility or relative mobility of a node.

The BoT type request is defined by the properties of the tasks and their values. For the DAG type of jobs we identify two additional characteristics: the granularity (g) and Communication to Computation Ratio (CCR). These two metrics for DAGs comparison are different measure for the computation/communication ratio. A DAG where the computation is dominant is *coarse-grained* (or CPU-bound). In a *fine-grained* DAG, the communication dominates the computation (IO-bound).

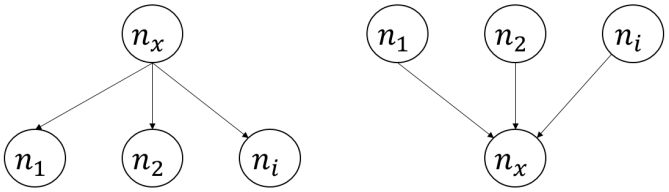

Figure 1: Join and Fork structure in a DAG

For the granularity g of a DAG $\mathcal{G}$ we use the definition introduced in [13]. We consider the structure of a *fork* associated to node n_x ($\mathcal{F}_x$) and a *join* associated to node n_x ($\mathcal{J}_x$) as shown in Figure 1. We define the granularity of a join, a fork and of a DAG as follows:

$$g(\mathcal{F}_x) := \frac{\min_{n_i \in V}\{w(n_i)\}}{\max_{(n_x,n_i)\in E}\{c(n_x,n_i)\}} \quad (fork \quad granularity) \tag{4}$$

$$g(\mathcal{J}_x) := \frac{\min_{n_i \in V}\{w(n_i)\}}{\max_{(n_i,n_x)\in E}\{c(n_i,n_x)\}} \quad (join \quad granularity) \tag{5}$$

$$g(\mathcal{G}) := \min_{n_x \in V}(g(\mathcal{F}_x), g(\mathcal{J}_x)) \quad (DAG \quad granularity) \tag{6}$$

We defined two values for CCR: $CCR_s(\mathcal{G})$ represents the average edge weight divided by the average node weight and $CCR_m(\mathcal{G})$ uses the max function for the computation:

$$CCR_s(\mathcal{G}) := CCR_s(\mathcal{G}(V,E,w,c)) = \frac{|V| * \sum_{(x,i)\in E} c(x,i)}{|E| * \sum_{x\in V} w(x)}, \tag{7}$$

$$CCR_m(\mathcal{G}) := CCR_m(\mathcal{G}(V,E,w,c)) = \frac{\max_{(x,i)\in E}\{c(x,i)\}}{\max_{x\in V}\{w(x)\}}. \tag{8}$$

For a coarse-grained DAG $\mathcal{G}$, $CCR(\mathcal{G}) > 1$ and $g(\mathcal{G}) > 1$. In the case of fine-grained ones, $CCR(\mathcal{G}) < 1$ and $g(\mathcal{G}) < 1$. At last, both metrics have values ≈ 1 for mixed DAGs.

A *Resource* is defined by a set of the following properties:

$$R = (P_1^R, P_2^R), \tag{9}$$

where

- P_1^R is the processing speed defined as the amount time in which an atomic operation is executed;
- P_2^R is the IO speed: the time needed to read or write a unit of data from or to the disk.

A set of resources is defined as:

$$\mathcal{R} = \left\{R_i, 1 \le i \le m | R_i = (P_1^{R_i}, P_2^{R_i})\right\}. \tag{10}$$

3.2. Proposed Scheduling Algorithm

Machine Learning (ML) represents a set of techniques used to create prediction models that are trained based on historical data and may produce valuable outputs for new input data [3]. The ML algorithms describe either supervised or unsupervised learning. In the case of supervised learning the algorithm learns to predict outcomes for new input data sets other than the training data. The unsupervised learning does not focus on predicting output, but to classify data using different features. In our approach we apply a supervised learning technique. The underlying motivation is to select a suited scheduling algorithm for a workload or a workflow, based on a prediction model trained with historical data an the features of the new request. In ML specific terms, the training data set is represented by a large number of requests (workloads modeled as BoT or workflows modeled as DAGs). Each request is characterized by several features like type, number of tasks, I/O rates, computation requirements, communication cost. Also, each request in the training data is labeled with the scheduling algorithm that provided best results as the measured outcome. The supervised ML contains a large number of algorithms: decision trees, regression, neural networks, state vector machines, kernel based techniques or instance based methods.

In this paper we propose a scheduling algorithm for asymptotic scheduling, based on a ML-Box tool. The algorithm is resumed as follows (see Algorithm 1). When a *request* arrives, the scheduler has to decide how to schedule the *tasks* on the available *resources*. The decision is taken using a MLBox tool: the scheduler provides as input for the MLBox the metadata of the request (the properties we defined above) and expects as output the scheduling algorithm that should be used, one of the traditional algorithms the system 'knows about'. In the initialization phase of the system, the underlying machine learning algorithm of the MLBox has to be trained, using a large number of requests with the best suited algorithm associated. In the training step, the ML algorithm build a model that will be used to decide what algorithm to use for new requests.

Algorithm 1 Scheduling Algorithm using ML.

```
1: procedure ML_SCHEDULING(Request, Resources, TrainingFile)
2:     MLBox.Train(TrainingFile);
3:     while true do
4:         Metadata = MLBox.Read_and_Profile(Request);
5:         Heuristic_Name = MLBox.Forecast(Metadata);
6:         Schedule_Request(Request, Resources, Heuristic_Name);
7:     end while
8: end procedure
```

Next we detail each entity used by this algorithm's micro-system (see Figure 2), as stated in the previous brief description: *MLBox* and *Scheduler*.

- *MLBox* is used in the scheduling process to select which heuristic should be applied for the new received request. It is based on a NN approach with multiple inputs and multiple outputs. The input is the metadata of the request: several properties defined above like granularity and CCR for a DAG or average IO or CPU requirements for a BoT request. It uses a set of requests and associated heuristics as training data. The heuristics in the training data have to be implemented in the *Scheduler*, otherwise the entry will be ignored.

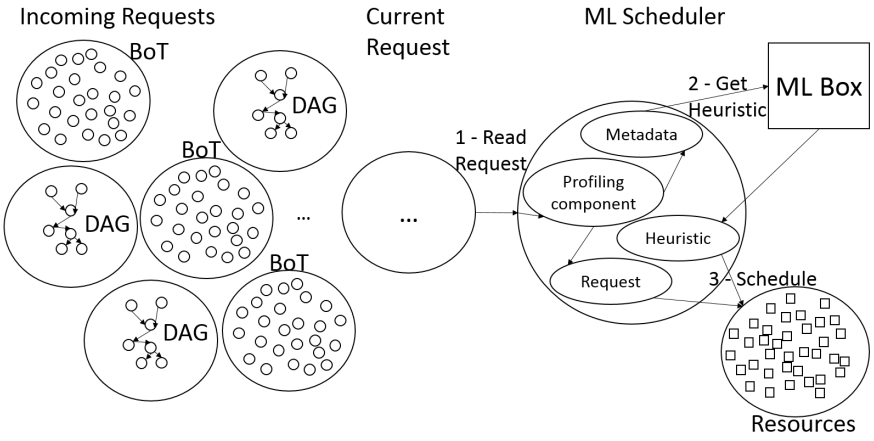

Figure 2: ML based Scheduling Flow.

- *Scheduler* receives the execution requests and has to assign them to the resources. It implements multiple heuristics. For BoT requests we chose four algorithms: Shortest Job First (SJF), Earliest Deadline First (EDF), Min-Max and Longest Job to Faster Resource - Shortest Job to Faster Resource (LJFR-SJFR) and four for DAGs: Modified Critical Path (MCP), Earliest Time First (ETF), Mobility Directed (MD) and Dynamic Critical Path (DCP). When *Scheduler* reads the request, it also 'profiles' - extracts the metadata, feed the metadata obtained as input for the MLBox, gets the scheduling algorithm to be applied and executes it (step 4 in Algorithm 1).

3.3. Inside the MLBox

MLBox component has a key role in our scheduling algorithm: it provides the best suited heuristic for a new request based on the model of the underlying NN. The ML technique has two main phases: the training phase and the actual prediction phase. In the training phase, the NN model parameters are computed. The prediction for new requests is performed using the initially established model.

Our MLBox uses two independent NNs, one for the requests of type BoT and one for the DAGs. The two NNs have a slightly different structure, as respect to the number of inputs, they will have different models because the array of weights for each node will vary and they will be trained in parallel in the initialization phase, as the training data contains both BoTs and DAGs. The MLBox will have a distinct component (switch) to dispatch each request to the corresponding NN, both during training and prediction phases, based on the request type.

The general rule to decide the configuration of a NN is not defined, nevertheless there are some guidelines to be followed. The input and output layers should have as many neurons as the number of inputs and outputs of the problem. Regarding the number of hidden layers, we can use no hidden layer if the input/output dependence is linear, or one or more hidden layers for more complex dependencies patterns. Generally, one hidden layer is enough to give good performance for almost all cases (hidden layers would mean training models that are more complex). The size of the hidden layer may be usually between the size of the input layer and the size of the output layer (different factors may apply here). The model of the proposed NNs contains three layers:

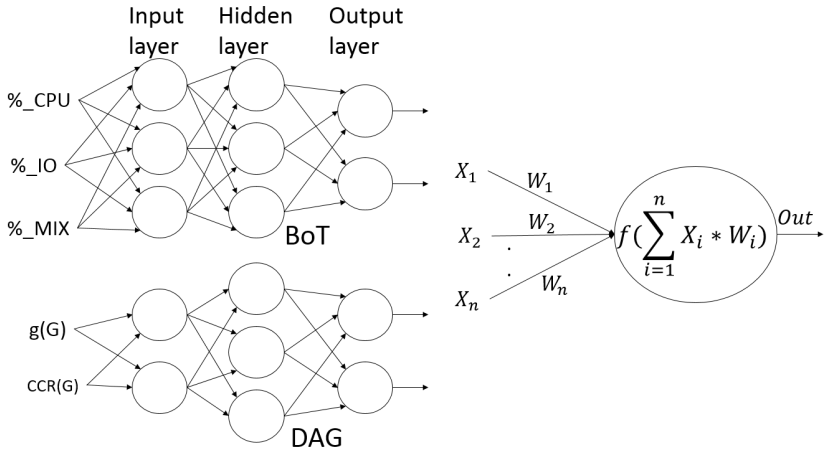

Figure 3: NN models for BoTs and DAGs, and detailed parameters for a single neuron.

- *Input layer* contains as many neurons as the inputs provided;
- One *Hidden layer*, that contains three neurons in both cases (in our model we have 3 neurons in the input layer and 2 neurons for the output layer, so we chose a hidden layer with 3 neurons);
- *Output layer* contains two neurons, which values can be 0/1. The two output values encode one of the four possible heuristics of each type of requests. If we add multiple heuristics, we also have to increase the number of outputs.

The *NN model for BoT* has three input values: the percentage of IO-bound tasks (%_IO), the percentage of CPU-bound tasks (%_CPU) and the percentage of mixed tasks (%_MIX - both IO and CPU bound). As for a DAG $\mathcal{G}$, we provide as input values, the granularity $g(\mathcal{G})$, and the Communication to Computation Ratio $CCR_s(\mathcal{G})$.

The scope of the training phase is to compute the parameters (weights) for each neuron in the model, for both network models proposed. As shown in Figure 3, each neuron is defined by an array of weights corresponding to each of its input values. The output of the neuron is represented by the result of an activation function applied to the weighted sum of input values. The main steps in the training phase:

- Initialize all weights with random values;
- Iterate through the training data as follows: if the generated output is different than in the data set, update weights using back propagation, and apply the new weights for the same set of inputs. This is done until we get the correct output.

3.4. Inside the Scheduler

The *Scheduler* component has multiple roles in our algorithm. First, it profiles a request as it reads it, then it calls the MLBox to get the appropriate heuristic for the job and finally executes the selected scheduling heuristic. The metadata that is computed in the profiling phase for a given request, has the following elements:

- the *Type* is the same as the type of the request: BoT or DAG,
- a set of properties, depending on the *type*:
  - BoT: %_IO, %_CPU and %_MIX;
  - DAG: $g(\mathcal{G})$, $CCR_s(\mathcal{G})$.

For BoT requests, we consider two very simple algorithms, Shortest Job First (SJF) and Earliest Deadline First (EDF) and two more complex heuristics: Min-Max and Longest Job to Faster Resource - Shortest Job to Faster Resource (LJFR-SJFR). Both SJF and EDF are static scheduling algorithms (see Algorithm 2). The tasks are sorted using a defined criteria: descending deadline (P_4^T) for EDF or ascending "length" (smallest processing time P_1^T) for SJF and after, each task is scheduled on a random available resource.

Algorithm 2 Shortest Job First and Earliest Deadline First.

procedure SJF_EDF($tasks, resources$)
  Sort tasks ascending using P_1^T for SJF **OR**
  Sort tasks descending using P_4^T for EDF;
  while $tasks \neq \phi$ **do**
    if $anyResourceAvailable(resources) = true$ **then**
      $R \leftarrow getRandomResourceAvailable(resources)$;
      $T \leftarrow popTask(tasks)$;
      execute T on R;
    end if
  end while
end procedure

Algorithm 3 Min-Max algorithm.

procedure MIN-MAX($Tasks, Resources$)
  $\mathcal{U}$ is the set of unmapped tasks
  while $\mathcal{U} \neq \phi$ **do**
    $\mathcal{Z} = \emptyset$
    for $\mathcal{T}_j \epsilon \mathcal{U}$ **do**
      $C_{ij} = \mathcal{W}_i + \mathcal{E}_{ij}; \quad \forall \mathcal{R}_i \epsilon Resources$
      $C_{xj} = \min C_{ij}, \quad \mathcal{E}_{hj} = \min \mathcal{E}_{ij}; \quad \forall \mathcal{R}_i \epsilon Resources$
      $\mathcal{Z} = \mathcal{Z} \cup \frac{C_{xj}}{\mathcal{E}_{hj}}$;
    end for
    $\mathcal{K}_{qp} = \max \mathcal{K}_{ij}; \quad \forall \mathcal{K}_{ij} \epsilon \mathcal{Z}$
    Schedule $\mathcal{T}_p$ on $\mathcal{R}_q$;
    $\mathcal{U} = \mathcal{U} - \mathcal{T}_p$
  end while
end procedure

On the other hand LJFR-SJFR and Min-Max use more complex metrics that depend on the resources, and the resource load (see Algorithm 3); these are dynamic scheduling algorithms. Both algorithms use a common metric, the minimum completion time (MCT).

- LJFR-SJFR uses two metrics, MCT (Longest Job) and maximum CT (Shortest Job) alternatively. For each metric a resource is selected. The longest jobs are scheduled on the selected resource for some initial steps. Next, assign alternatively longest/shortest jobs;
- Min-Max adds another metric: the minimum execution time (MET) combined with MCT. Two resources are selected for the same tasks, one for MCT (R_x), and one associated with MET (R_h). The task with maximum ratio $\frac{Exec_{R_x}}{Exec_{R_h}}$ will be scheduled on resource R_x.

For DAG requests we implemented Modified Critical Path (MCP) and Earliest Time First (ETF) - they are simple algorithms with a low complexity, giving good enough results in some cases in a shorter time. Two more complex algorithms are Mobility Directed (MD) and Dynamic Critical Path (DCP) which generate good schedules also for DAGs with increased numbers of forks and joins.

- The MCP algorithm based on lists has two disjoint phases: assign priority of tasks using ALAP and selection of resources for current task computing the earliest start time.
- In the case of ETF the goal is to keep the processors as busy as possible. It computes, at each step, the earliest start times of all ready nodes and selects the one with the shortest start time and the associated resource (see Algorithm 4).

Algorithm 4 Earliest Time First (ETF) algorithm.

```
1:  procedure ETF(G, R)
2:      RN = entry nodes from G;                          ▷ the pool of ready nodes;
3:      repeat
4:          for each node n_i ∈ RN do
5:              Calculate the earliest-start-time(n_i) on each resource in R;
6:          end for
7:          Pick the (n_i, R) that gives the earliest time;
8:                                                  ▷ using the non-insertion approach;
9:          Ties are broken by selecting the node n_i with a higher static b − level(n_i);
10:         Schedule the node to the corresponding resource;
11:         Add the newly ready nodes to the RN;
12:     until all nodes are scheduled
13: end procedure
```

- The MD algorithm defines additional properties for a node in the DAG: the *mobility* of a node (or moving range), the *moving interval* and the *relative mobility*. This algorithm is also a list-based one, in which the nodes are selected in the ascending order of the *relative mobility*. Then, the assignment on a resource is performed by checking that the moving interval and range for each task already scheduled on the resource fulfill a necessary and sufficient condition (Eq. 16 or Eq. 17). The current selected task, can be scheduled before already assigned tasks on the selected resource. After node n_j is scheduled on resource $\mathcal{R}_i$, the DAG is updated: set edges that connect n_j to other nodes scheduled on R_i to 0 or add edges of weight 0 between these nodes where they are missing. If any loops are created,

then schedule n_j on another resource.

$$\begin{aligned} \tau_S(n_i) &= ASAP(n_i); \\ \tau_L(n_i) &= ALAP(n_i); \\ \tau_F(n_i) &= \tau_L(n_i) + w(n_i); \end{aligned} \tag{11}$$

$$\mathcal{MR}(n_i) = [\tau_S(n_i), \tau_L(n_i)] \quad (moving \quad range) \tag{12}$$

$$\mathcal{MI}(n_i) = [\tau_S(n_i), \tau_F(n_i)] \quad (moving \quad interval) \tag{13}$$

$$\mathcal{M}(n_i) = \tau_L(n_i) - \tau_S(n_i) \quad (mobility) \tag{14}$$

$$\mathcal{M}_r(n_i) = \frac{\mathcal{M}(n_i)}{w(n_i)} \quad (relative \quad mobility) \tag{15}$$

Consider the set $\{n_{m_1} n_{m_l}\}$ of nodes already scheduled on resource $\mathcal{R}_i$. The necessary and sufficient condition to schedule node n_j on resource $\mathcal{R}_i$is:

$$\mathcal{MI}(n_j) \cap \mathcal{MI}(n_{m_x}) = \phi, \quad \forall n_{m_x} \tag{16}$$

$$\exists k: \quad w(n_j) \leq \min(\tau_F(n_j), \tau_L(n_{m_k})) - \max(\tau_S(n_j), \tau_S(n_{m_{k-1}}) + w(n_{m_{k-1}})) \tag{17}$$

where $\mathcal{MI}(n_j) \cap \mathcal{MI}(n_{m_k}) \neq \phi$.

- HLFET was implemented as an alternative to MD (if the algorithm takes too much to schedule the resources) - it prioritizes tasks using the static b-level attribute.
- A composite attribute for tasks priority is used in the DLS algorithm. The dynamic level is the computed as the difference between the sl and the EST, for each task on each resource. At each iteration, the DL is computed for all ready nodes and the pair with the largest DL value is selected. The difference between DLS and ETF is that ETF always schedules the tasks in ascending order of the EST, when DLS uses for start using the descending order of the sl and towards the end uses the ascending order of EST.

4. Evaluation Methodology and Results

Current scheduling research challenges are generally caused by the large data sets gathered, analyzed, processed, represented and stored. Also for our test case scenarios, we chose both independent tasks and tasks with dependencies (workflows) as the current applications generate these both types of jobs. Moreover, the BigData applications may generate CPU intensive (they process a large amount of data) tasks, I/O intensive (they access remote data) or both, so we considered them when generating tasks for DAGs and BoT.

4.1. Implementation Details

For the validation of the scheduling algorithm we presented in this paper, we chose to implement the described model in CloudSim. The implementation of the MLBox Scheduler consists in three main entities: *Main*, *Scheduler* and *MLBox*. They are associated with three of the packages of the main structure (as shown in Figure 4). The fourth package contains the model classes, which define the additional types needed in our algorithm. The *Main* component handles the incoming requests: reads a request, profiles it, gets the information about the scheduling algorithm to be applied and submits the request to the *Scheduler* component. The scheduler executes the heuristic associated with the request and submits the tasks to the resources.

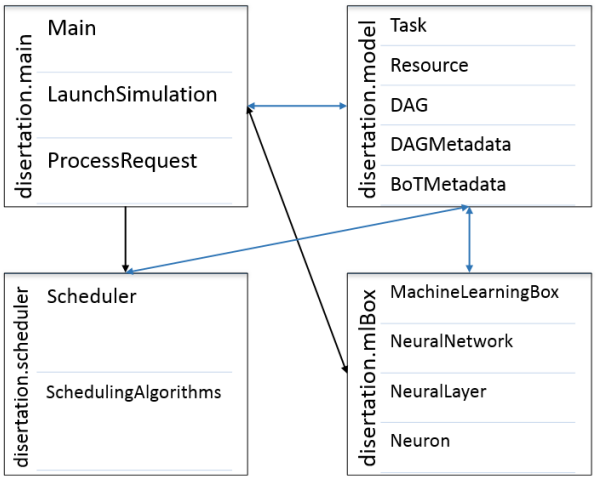

Figure 4: MLBox Scheduler Main Package Structure.

The main steps for realizing our experiments were: generate training data for the MLBox, train the ML algorithm and get the model both for predicting the scheduling algorithm for BoTs and for DAGs, use the MLBox to get the optimal scheduling algorithm for a given request.

4.2. Experiments Set-Up

1. Hardware configuration
   In our experiments we considered a fixed configuration for the Cloud resource. We used a homogeneous Cloud due to the fact that the DAG scheduling traditional algorithms rely on identical processing elements. The Cloud infrastructure we used contains 10 identical virtual machines. They are deployed on multiple hosts (having enough memory and processors elements) to accommodate the VMs. For a single VM we have the following characteristics:

   - *Processing Capacity* 5000 MIPS;
   - *RAM*: 1024;
   - *Processing Elements* number: 2 PEs (PE is the equivalent of a Core)

2. DAG generation
   We generated the DAGs we used for testing with a program written in C, that may be configured with some general parameters (like maximum and minimum nodes on a level, maximum and minimum levels), the probability to have and edge between two random nodes (edges are generated always from the higher levels to the lower levels), and the maximum value for a node weight and for an edge weight. Based on this configuration, the DAGs parameters are generated according with the distribution presented in Figure 5.

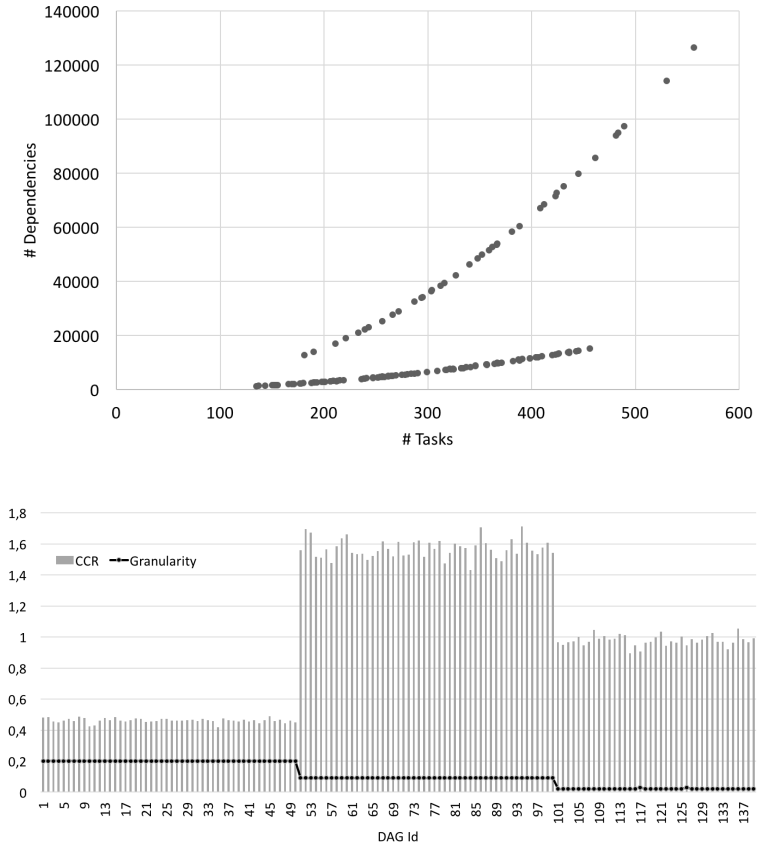

Figure 5: Number of tasks in the generated DAGs and properties of generated DAGs (CCR and granularity).

We generated 140 DAGs. The number of tasks in each DAG varied too, because the number of nodes on each layer and number of layers are generated randomly. In our data set, we had between 200 and 600 tasks per DAG.
As a more detailed description of the DAG structure: we generated DAGs in the three categories identified in the previous sections: coarse-grained, fine-grained and mixed. To achieve this, we varied the maximum value for nodes and edges weight. For the generated graphs, we did not manage to get a granularity higher than one, but the CCR values were lower, greater and around 1.

4.3. Numerical Results

For evaluating the algorithms, we considered as metric the makespan, so the algorithm with minimum makespan is selected as the best for the request (see Figure 6). As a short summary, our algorithm receives a request, creates a profile (generates the metatdata), gets the best algorithm from the MLBox, and the executes that algorithm.

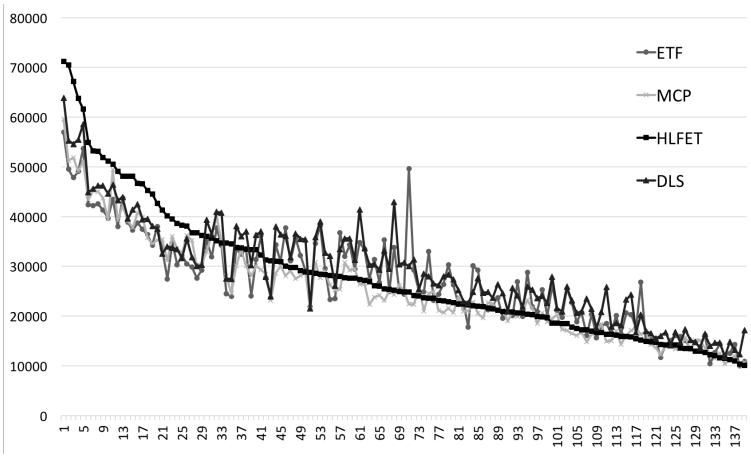

Figure 6: Makespan comparison.

To create the training data for the NN, we ran all scheduling algorithms for the generated DAGs, chose the best algorithm and added an entry in the training data:

- Test Inputs: $g(\mathcal{G})$ and $CCR(\mathcal{G})$;
- Target Outputs: b_1, b_2 (the codification of the scheduling algorithm: '00' - ETF, '01'-MCP, '10'-HLFET, '11'-DLS).

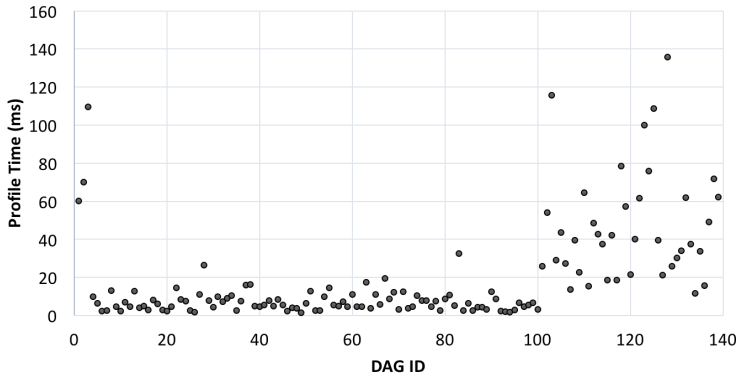

Figure 7: Profiling Delay.

In terms of evaluating the performance for the algorithm we have to take into account the following: the makespan and mean flowtime of the generated schedule, together with the scheduling

execution time are the same as for the best algorithm for the current request. The additional step we proposed is the decision of the algorithm divided in two: profiling and predict using the MLBox (see Figure 7).

We analyzed the delay introduced by the profiling phase, by measuring its execution time in the scheduling process for all DAGs in the training data set. The delays for BoT applications are comparable, and depend on the size of the request (in terms of number of nodes).

The MLBox training takes place in the initialization of the system and does not influence the actual scheduling speed. For the prediction delay, this is not important as size, because the number of outputs and inputs is reduced, the NN has a simple layout and the model is already defined.

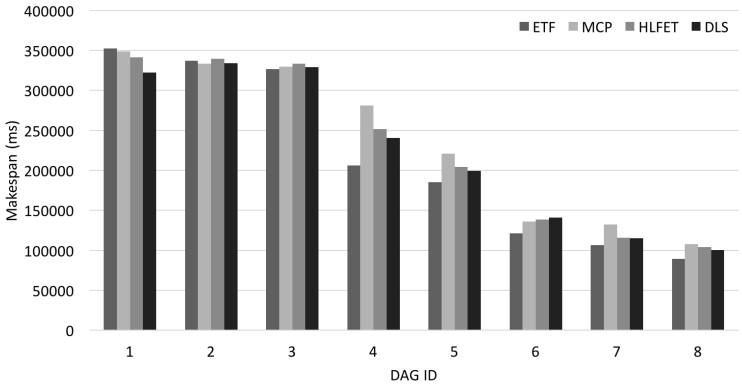

Figure 8: Test DAGs results.

For testing of the ML algorithm, we used a small set of particular DAGs inspired from the ones used to model different types of workflows (Laplace graph, Balanced graph or LU graph). The evaluation of selected scheduling algorithm is presented in Figure 8. The algorithm chose the best technique in most of the cases, we tested this by running all algorithms for the DAGs, compare the makespan and check with the result of the NN prediction (see Table 1).

DAG Id	Actual Optimum	Prediction
1	MCP	ETF
2	ETF	ETF
3	ETF	ETF
4	ETF	DLS
5	ETF	ETF
6	ETF	ETF
7	ETF	ETF
8	DLS	ETF

Table 1: Actual vs. Prediction.

4.4. Interpretation of results

Our previous experiments proved that the algorithm we implemented performs well for scheduling requests in a Cloud simulation environment. The decision part does not delay the overall algorithm speed due to the simple (but efficient) architecture of the NN, and the fact that the weights needed for prediction are determined at system start up.

Even though the profiling introduces a small delay, the best scheduling algorithm is chosen, so the makespan is reduced and costs of execution. This system it is not suited for real-time systems, that need a very large speed, but is a good choice for system that face high work loads periods, receive large data sets that can be delayed. If no delay is accepted, a default scheduling algorithm could be applied.

5. Conclusions

In this paper we proposed a scheduling algorithm based on a Machine Learning Box for asymptotic requests: we described the model we built, then the implementation in a simulation environment of the novel algorithm and of 8 heuristics both for BoTs and DAGs and last, we performed some tests for comparing the makespan and mean flowtime both for large BoT and DAG requests. The algorithm we proposed in the paper targets the Big Data applications challenges: requests with very large number of tasks, increased CPU and IO requirements and requests with variable representation of the tasks (BoT and DAG). The algorithm performs well, as the ML algorithm for predicting the best suited heuristic has a very good speed as the model is already computed, and the prediction is based on some aggregated characteristics of the requests. The overall scheduling process is not delayed by this prediction step. The ML training step is not considered as part of the scheduling process, and it is performed at the initialization of the system, based on input training data from a persistent storage, a model is determined for each of the task types (BoT and DAG) and the model is used for further predictions. We validated both the Machine Learning algorithm and the proposed scheduling approach. We performed tests for to check the model of the MLBox and analyzed the performance comparison of our algorithm and the traditional heuristics.

As future work, the proposed model and actual scheduling algorithm may be enhanced with several improvements. The development directions we identified are: consider the resource heterogeneity and adapt the DAG heuristics that consider that the processing elements are identical, consider the idea of dynamic resource provisioning and propose an integration of the system in a real-life Cloud platform. *Resource heterogeneity* should be involved in the scheduling algorithm because modern Cloud systems use heterogeneous resources. The Cloud heterogeneity has some advantages like easily switch from one architecture to another based on requirements and the user groups. *Resource provisioning* is used frequently by Cloud providers: resources are added to the system on peak utilization periods or removed when the work load decreases. In this transition period, the system should efficiently execute incoming jobs, even though the system is changing.

The algorithm could be also tested and validated in real Cloud environments, like Hadoop, Amazon's EC2 or BlueMix from IBM. Int this way the experiments would reveal real-life situations. The proposed steps for achieving this goal do not depend on the selected platform (just the actual scheduling step) and are resumed as follows:

- deploy the MLBox as a stand alone application on the same system as the scheduler and provide the training data;

- deploy the Main application and design a communication protocol between the Main application and the MLBox and *Scheduler*;
- extend the default scheduler of the system with the required heuristics, and the new scheduling algorithm. This step depends on how the scheduling mechanism is implemented in the selected platform.

Acknowledgment

The research presented in this paper is supported by projects: *DataWay*: Real-time Data Processing Platform for Smart Cities: Making sense of Big Data - PN-II-RU-TE-2014-4-2731; *MobiWay*: Mobility Beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow - PN-II-PT-PCCA-2013-4-0321; *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *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.

We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

References

[1] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, R. Buyya, Big data computing and clouds: Trends and future directions, Journal of Parallel and Distributed Computing 79 (2015) 3–15.

[2] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, M.-H. Su, Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand, in: SPIE Astronomical Telescopes+ Instrumentation, International Society for Optics and Photonics, 2004, pp. 221–232.

[3] C. M. Bishop, Pattern recognition and machine learning, springer, 2006.

[4] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, L. Zhou, Apollo: scalable and coordinated scheduling for cloud-scale computing, in: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), 2014, pp. 285–300.

[5] G. Brumfiel, High-energy physics: Down the petabyte highway, Nature News 469 (7330) (2011) 282–283.

[6] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.

[7] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Modium, K. Singh, J. Suh, J. P. Walters, Heterogeneous cloud computing, in: Cluster Computing (CLUSTER), 2011 IEEE International Conference on, IEEE, 2011, pp. 378–385.

[8] R. F. de Mello, L. J. Senger, L. T. Yang, A routing load balancing policy for grid computing environments, in: 20th International Conference on Advanced Information Networking and Applications-Volume 1 (AINA'06), vol. 1, IEEE, 2006, pp. 6–pp.

[9] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, et al., Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Scientific Programming 13 (3) (2005) 219–237.

[10] C. Delimitrou, C. Kozyrakis, Paragon: Qos-aware scheduling for heterogeneous datacenters, in: ACM SIGPLAN Notices, vol. 48, ACM, 2013, pp. 77–88.

[11] J. E. Drew, R. Greimel, M. J. Irwin, A. Aungwerojwit, M. J. Barlow, R. L. Corradi, J. J. Drake, B. T. Gänsicke, P. Groot, A. Hales, et al., The int photometric hα survey of the northern galactic plane (iphas), Monthly Notices of the Royal Astronomical Society 362 (3) (2005) 753–776.

[12] U. Fiore, F. Palmieri, A. Castiglione, A. De Santis, A cluster-based data-centric model for network-aware task scheduling in distributed systems, International Journal of Parallel Programming 42 (5) (2014) 755–775.

[13] A. Gerasoulis, T. Yang, On the granularity and clustering of directed acyclic task graphs, Parallel and Distributed Systems, IEEE Transactions on 4 (6) (1993) 686–701.

[14] A. Goder, A. Spiridonov, Y. Wang, Bistro: Scheduling data-parallel jobs against live production systems, in: 2015 USENIX Annual Technical Conference (USENIX ATC 15), 2015, pp. 459–471.

[15] A. V. Gritsenko, N. G. Demurchev, V. V. Kopytov, A. O. Shulgin, Decomposition analysis and machine learning in a workflow-forecast approach to the task scheduling problem for high-loaded distributed systems, Modern Applied Science 9 (5) (2015) p38.
[16] J. O. Gutierrez-Garcia, K. M. Sim, A family of heuristics for agent-based elastic cloud bag-of-tasks concurrent scheduling, Future Generation Computer Systems 29 (7) (2013) 1682–1699.
[17] T. Hagras, J. Janeček, Static vs. dynamic list-scheduling performance comparison, Acta Polytechnica 43 (6).
[18] L. He, H. Zhu, S. A. Jarvis, Developing graph-based co-scheduling algorithms on multicore computers, IEEE Transactions on Parallel and Distributed Systems 27 (6) (2016) 1617–1632.
[19] A. P. Heath, M. Greenway, R. Powell, J. Spring, R. Suarez, D. Hanley, C. Bandlamudi, M. E. McNerney, K. P. White, R. L. Grossman, Bionimbus: a cloud for managing, analyzing and sharing large genomics datasets, Journal of the American Medical Informatics Association 21 (6) (2014) 969–975.
[20] J. Huang, H. Jin, X. Xie, Q. Zhang, Using narx neural network based load prediction to improve scheduling decision in grid environments, in: Natural Computation, 2007. ICNC 2007. Third International Conference on, vol. 5, IEEE, 2007, pp. 718–724.
[21] H. Izakian, A. Abraham, V. Snášel, Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments, Neural Network World 19 (6) (2009) 695–710.
[22] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Computing Surveys (CSUR) 31 (4) (1999) 406–471.
[23] Y.-K. Kwok, L. Ahmad, Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors, Parallel and Distributed Systems, IEEE Transactions on 7 (5) (1996) 506–521.
[24] M. Lin, L. T. Yang, Hybrid genetic algorithms for scheduling partially ordered tasks in a multi-processor environment, in: Real-Time Computing Systems and Applications, 1999. RTCSA'99. Sixth International Conference on, IEEE, 1999, pp. 382–387.
[25] H. Liu, A. Abraham, V. Snášel, S. McLoone, Swarm scheduling approaches for work-flow applications with security constraints in distributed data-intensive computing environments, Information Sciences 192 (2012) 228–243.
[26] R. Mian, P. Martin, J. L. Vazquez-Poletti, Provisioning data analytic workloads in a cloud, Future Generation Computer Systems 29 (6) (2013) 1452–1458.
[27] F. Palmieri, U. Fiore, S. Ricciardi, A. Castiglione, Grasp-based resource re-optimization for effective big data access in federated clouds, Future Generation Computer Systems 54 (2016) 168–179.
[28] K. Paranjape, S. Hebert, B. Masson, Heterogeneous computing in the cloud: Crunching big data and democratizing hpc access for the life sciences, Intel White Paper.
[29] F. Pop, C. Dobre, V. Cristea, Genetic algorithm for dag scheduling in grid environments, in: Intelligent Computer Communication and Processing, 2009. ICCP 2009. IEEE 5th International Conference on, IEEE, 2009, pp. 299–305.
[30] I. Raicu, Many-task computing: bridging the gap between high-throughput computing and high-performance computing, ProQuest, 2009.
[31] I. Raicu, I. Foster, Y. Zhao, A. Szalay, P. Little, C. M. Moretti, A. Chaudhary, D. Thain, Towards data intensive many-task computing, Data Intensive Distributed Computing: Challenges and Solutions for Largescale Information Management 13 (3) (2012) 28–73.
[32] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, M. Samidi, Scheduling data-intensiveworkflows onto storage-constrained distributed resources, in: Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on, IEEE, 2007, pp. 401–409.
[33] A. Sfrent, F. Pop, Asymptotic scheduling for many task computing in big data platforms, Information Sciences.
[34] M. Sheikhalishahi, R. M. Wallace, L. Grandinetti, J. L. Vazquez-Poletti, F. Guerriero, A multi-dimensional job scheduling, Future Generation Computer Systems 54 (2016) 123–131.
[35] G. Singh, M.-H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, G. Mehta, Workflow task clustering for best effort systems with pegasus, in: Proceedings of the 15th ACM Mardi Gras Conference: From Lightweight Mash-ups to Lambda Grids: Understanding the Spectrum of Distributed Computing Requirements, Applications, Tools, Infrastructures, Interoperability, and the Incremental Adoption of Key Capabilities, MG '08, 2008, pp. 9:1–9:8.
[36] S. Suneja, E. Baron, R. E DE LARA, Accelerating the cloud with heterogeneous computing, Proceedings of HotCloud (2011) 1–5.
[37] P. Vagata, K. Wilfong, Scaling the facebook data warehouse to 300 pb (2014).
[38] N. Vasić, D. Novaković, S. Miučin, D. Kostić, R. Bianchini, Dejavu: accelerating resource allocation in virtualized environments, in: ACM SIGARCH computer architecture news, vol. 40, ACM, 2012, pp. 423–436.
[39] M. Vasile, F. Pop, R. Tutueanu, V. Cristea, Hysarc2: Hybrid scheduling algorithm based on resource clustering in cloud environments, in: Algorithms and Architectures for Parallel Processing - 13th International Conference, ICA3PP 2013, Vietri sul Mare, Italy, December 18-20, 2013, Proceedings, Part I, 2013, pp. 416–425.
[40] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea, J. Kolodziej, Resource-aware hybrid scheduling algorithm in

heterogeneous distributed computing, Future Gener. Comput. Syst. 51 (C) (2015) 61–71.
[41] M.-Y. Wu, D. D. Gajski, Hypertool: A programming aid for message-passing systems, IEEE Transactions on Parallel & Distributed Systems (3) (1990) 330–343.
[42] Y. Xu, K. Li, L. He, L. Zhang, K. Li, A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems, IEEE Transactions on Parallel and Distributed Systems 26 (12) (2015) 3208–3222.
[43] Y. Xu, K. Li, J. Hu, K. Li, A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues, Information Sciences 270 (2014) 255–287.
[44] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, K. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, Information Sciences 319 (2015) 113–131.
[45] D. Zissis, D. Lekkas, Addressing cloud computing security issues, Future Generation computer systems 28 (3) (2012) 583–592.