2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems

SLA-aware Scheduling of Map-Reduce Applications on Public Clouds

Xuezhi Zeng[1], Saurabh Garg[2] , Zhenyu Wen [3], Peter Strazdins[1] , Lizhe Wang[4], Rajiv Ranjan[5]
[1] Department of Computer Science, Australian National University, Australia
[2] Discipline of ICT, University of Tasmania, Australia
[3] School of Informatics, The University of Edinburg, United Kingdom
[4] Institute of remote sensing, Chinese Academy of Sciences
[5] School of Computing Science, Newcastle University, United Kingdom

***Abstract*—The recent need of processing BigData has led to the development of several Map-Reduce applications for efficient large scale processing. Due to on-demand availability of large computing resources, Public Clouds have become a natural host of these Map-Reduce applications. In this case, users need to decide which resources they need to rent to run their Map-Reduce cluster other than deployment or scheduling of map-reduce tasks itself. This is not a trivial task particularly when users may have performance constraints such as deadline and have several Cloud product types to choose with intention of not spending much money. Even though there are several existing scheduling systems, however most of them are not developed to manage the scheduling of Map-Reduce applications. That is, they do not consider things like the number of map and reduce tasks and slots per VM. This paper proposes a novel greedy scheduling algorithm (MASA) that considers the users constraints in order to minimize cost of renting Cloud resources while considering the user's budget and deadline constraints. The simulation results show 25-60% reduction cost in comparison to current methods by using our proposed algorithm.**

I. INTRODUCTION

Efficiently and effectively processing large volumes of BigData has emerged as a predominant challenge in many emerging application domains including (but not limited to) enterprise computing, smart cities, remote healthcare, high-energy physics, bio-informatics, and astronomy. For example, in the past enterprises have developed offline Business Intelligence (BI) for strategic decision making via analysis (done by experts) of historical data and decision-making cycles in the last weeks or months [1]. With the push towards more automation for faster business strategy adaptation, enterprises are facing the need to develop next generation BI systems that can support data-driven decision making. For example, on-line retail companies are required to analyze click stream data and up-to-the minute inventory status for offering dynamically priced and customized product bundles. Similarly, banks are looking to detect and react to frauds in based on analyzing transactional data. On the other hand, cities are evolving into smart cities by fusing and analyzing data from several sources (e.g., traffic cameras, social media, remote sensing data, GPS data).

Public cloud providers such as Amazon Web Services have started to offer on-demand Hadoop clusters (PaaS), referred to as Elastic Map-Reduce, on its EC2 datacentres (IaaS) on pay-as-you-go basis [2]. However, current scheduling techniques and systems for deploying Hadoop clusters [3] on public IaaS clouds are incapable of supporting SLA-driven data processing application management. Important SLA constraints include: (i) Deadline: upper bound on the time finishing the data processing task and (ii) Budget: upper bound on the monetary limit for finishing the data processing task. In the current practice, public cloud providers require users (Map-Reduce application administrators) to manually decide the mix and type of IaaS resources they need as part of their Hadoop cluster for finishing the analytic task over their BigData within SLA constraints.

Research Problem. Clearly, it is impossible to resolve such dependency between IaaS-level hardware configurations, deployment plan for Hadoops PaaS-level software components and SLA constraints manually. In particular, the hard challenge is to flexibly select IaaS configurations (I/O capacity, RAM, VM speed, local storage, cost) for scheduling PaaS-level Hadoops software components (such as number of Map tasks, number of Reduce tasks; Map Slots per VM, Reduce Slots for VM, Max RAM per slot) driven by SLA constraints (analyze 100GB of Tweets in 10 minutes subject to maximum budget of $100). The space of possible configurations for big data processing frameworks and hardware resource is very large, so computing an optimal solution is NP-complete, and therefore intractable given current technology.

The scheduling problem is further complicated by the fact that Map-Reduce application workload characteristics (e.g., data volume, priority, concurrency) and IaaS resource performance (e.g., availability, throughput, utilization) behavior fluctuate over time. Furthermore, as public cloud providers desire to maximize resource utilization and profit, they have mechanisms of dynamically consolidating other types of workload (e.g. web servers, video streaming, SQL/NoSQL query processing, stream processing) to the unused physical resources in the cluster which further adds to the complexity of dynamically managing clusters performance for meeting SLA constraints. In reality, the performance degradation depends on how noisy neighboring application workloads are. In most of the cases, it is likely that Map-Reduce applications will miss their deadline, which may result in financial losses based on analytic context. For example, delays in detecting fraudulent transaction may incur heavy losses to banks. On the other hand, delays in analyzing customer sentiments for products may lead to revenue loss for on-line retail companies.

Research Methodology and Contributions. The question

978-1-5090-4297-5/16 $31.00 © 2016 IEEE
DOI 10.1109/HPCC-SmartCity-DSS.2016.183

of SLA-aware scheduling of applications has been addressed previously in context of HPC, Grid, Cloud (at IaaS-layer), and Database research over the last 2 decades. Our methodology differentiates itself in following aspects. First of all, we present a mathematical model that enables holistic modeling of relationship between SLA parameters (e.g., budget and deadline) and Hadoop clusters configurations in terms of: (i) Big Data volume (ii) PaaS component configuration (number of mappers and number of reducers) and IaaS configuration (VM type, VM speed). Secondly, we develop a greedy heuristic-based scheduling algorithm that can pro-actively minimize the cost under user's constraint (budget and deadline), BigData workload (data volume, priority) and IaaS performance (e.g., availability, throughput, utilization) uncertainties. We extensively validate the performance of the SLA model and greedy scheduling algorithm in the IoTSim [4] simulator.

The rest of this paper is organized as follows. In Section 2, we discuss some related works. Section 3 presents the high level system scenario that is considered for scheduling Map-Reduce application(aka. Jobs) on Public Clouds. Section 4 discusses our mathematical model and its assumptions. In Section 5, we present our proposed scheduling algorithm. Section 6 presents evaluation of the performance of our proposed algorithm. In Section 7, we conclude the paper with future directions.

II. Related Work

While public clouds have evolved towards heterogeneous hardware configuration for differentiated processing power, I/O capacity, RAM size, network connectivity and network location, most existing Map-Reduce application scheduling platforms (Apache YARN, Apache Mesos, Apache Spark, Amazon Elastic Map-Reduce) are designed for homogeneous clusters of hardware resources (VM, Storage, and Network). These platforms expect Map-Reduce application administrators to determine the number and configuration of hardware resources to be allocated PaaS-level components (e.g., number of Map tasks, number of Reduce tasks; Map Slots per CPU, Reduce Slots for CPU, Max RAM per slot). Branded price calculators are available from public cloud providers (Amazon[5], Azure[6]) and academic projects (Cloudrado[7]) but they cannot recommend hardware configurations to be allocated to PaaS-level Hadoop components driven by SLA constraints.

There are several works on scheduling different applications on public cloud. Some have proposed algorithms to manage web applications, others for managing scientific applications and some on scientific workflow. Data and control flow dependencies in Map-Reduce applications are quite different from workflow as number of tasks in an application are not static as in traditional workflow but depends on data size. Thus, the existing algorithms that are proposed for scientific workflows cannot be applied in this scenario.

Lee et al.[8] proposed dynamically allocating public cloud resources to a Hadoop cluster based on a simple SLA constraint: minimize storage size. Kambatla et al. [9] proposed selecting the optimal set of public cloud resources for Hadoop cluster by developing and profiling hardware resource consumption statistics. Similarly, the authors of [10] propose selecting configurations of heterogeneous Amazon EC2 resources under various what-if scenarios (number of Map tasks, number of Reduce tasks, size and distribution of input data). However, none of these approaches considered deadline and budget SLA constraints while taking scheduling decisions. Some works such [11][12][13] proposed algorithms to minimize makespan of multiple map-reduce jobs on the same cluster where each job is competing for the resources, which is not the case of a Public Cloud. [14] shares a similar scenario with these papers, but the authors assume that map tasks and reduce tasks have the constraints in terms of monetary and execution time without any consideration of data transmission time. This assumption is not very realistic, because users are not able to set the specific constraints for each map or reduce task. In this paper, we only ask the users to set the constraints for the Map-Reduce application. Moreover, we consider the time of data transmission in our model which might significantly affects total makespan when the data size is huge.

Some public cloud providers such as Amazon Auto Scaler controllers implement reactive heuristics to scale Hadoop clusters based on CPU usage measurements. However, Auto Scaler warrants the administrator to statically define the CPU thresholds (no support for dynamic scaling based on optimizing SLA constraints) that triggers the scaling of cluster. There has been lot research work conducted in developing performance models of Map-Reduce applications on clouds. These performance models are based on machine learning [15], Markov and fast fourier transforms [16] or using wavelet techniques [17] and they provide predictions on execution (run) time of Map/Reduce tasks, input data volume, network I/O patterns, etc. Principal Component Analysis [18] has also been applied to learn performance model parameters of Map-Reduce applications. The progress made in Map-Reduce performance modeling is significant however, these models do not cater for SLA constraints while undertaking prediction decisions.

In summary, to best of our knowledge, it is the first work that model all the requirements of a map-reduce job (both computing and communication) and schedule them on Public Cloud in order to minimize their execution cost while considering SLA parameters such as Budget and Deadline.

III. System Model: Scheduling Framework

Current BigData processing platforms for deploying Map-Reduce applications can be divided into following layers:

(i) Platform as a Service (PaaS) level software frameworks such as Apache Hadoop [3] (an implementation of Google's Map-Reduce programming model [19]) that enable development of parallel data processing applications by exploiting capacity of large cluster of commodity computers. Frameworks such as Apache Hadoop hide the low-level distributed system management complexities by automatically taking care of activities including task

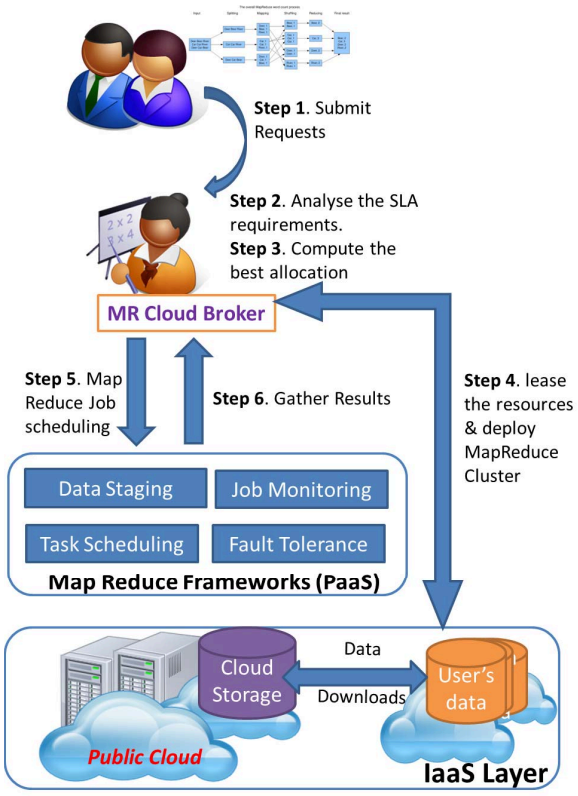


Fig. 1. Map Reduce Job Scheduling on Public Cloud

scheduling, data staging, fault management, inter-process communication, result collection.

(ii) Infrastructure as a Service (IaaS) that offers unlimited data storage and processing capabilities hosted in large datacentre farms.

Figure 1 shows our scheduling scenario. In our approach, users submit their requests to a MR (Map-Reduce) Cloud broker whose responsibility is to select appropriate IaaS datacentre services for deploying Hadoop Cluster on behalf of an user (e.g., data analyst) while meeting user-specified SLA constraints. The users' deployment request consist of details of map reduce application features including data size to process, deadline by which a user wants the job to be finished and a budget which she is willing to spend.

MR Cloud Broker has the similar responsibility as a typical Cloud broker, i.e. to interact with users, understand their requirements and schedule processing based on users' SLA constraints. The scheduling algorithm, models, and assumption for making this decision is discussed in the following sections. The broker will decide which type of Virtual Machines (VMs) to be utilized so that cost of execution can be minimized. It will also decide where (e.g., type of VM) each map and reduce task should execute. After analyzing the user's request, the MR broker deploys a Map-Reduce cluster after negotiating all the required IaaS services from the datacentre provider.

In the next section we will discuss the mathematical model and assumption that is considered as part of our scheduling approach.

IV. Mathematical Model and Assumptions

TABLE I
Notations

Symbols	Descriptions
MapReduce Workload	
J	a Map-Reduce Job or workload set
J_i	a Map-Reduce Job or workload instance
M_i	a set of map tasks $\in J_i$
R_i	a set of reduce tasks $\in J_i$
$Size_{map}(J_i)$	the input data size of map tasks $\in J_i$
$Size_{reduce}(J_i)$	the input data size of the reduce tasks $\in J_i$
VM Configuration	
VM	a set of VM types
VM_{mips}	set denoting MIPS rating of VMs
VM_{map}	upper limit on number of map tasks that can be mapped to VM
VM_{reduce}	upper limit on number of reduce tasks that can be mapped to VM
$VM_{bandwidth}$	VMs' network bandwidth
Y	the leasing cost of a Small VM type for a minute
vm_j	a VM instance j
makespan	
$TT(M_i, vm_j)$	network delay (transfer cost) in transferring input data of map tasks of J_i to vm_j
$TT(R_i, vm_j)$	network delay (transfer cost) in transferring input data of reduce tasks of J_i to vm_j
vm_j^{bw}	the network bandwidth of vm_j
$MIMT(M_i)$	J_i's aggregated millions instructions (MI) of map tasks
$TET(M_i, vm_j)$	the execution time of the map tasks of J_i
$MIRT(R_i)$	J_i's aggregated millions instructions (MI) of reduce tasks
$TET(R_i, vm_j)$	the execution time of the reduce tasks of J_i
$makespan(J_i, hVM)$	the total makespan of executing J_i over hVM VMs
$makespan_m(J_i, hVM')$	the makespan of executing map tasks of J_i over hVM' VMs
$makespan_r(J_i, hVM'')$	the total makespan of executing reduce tasks of J_i over hVM'' VMs
Monetary Cost	
$COST(J_i, hVM)$	the cost of execution J_i over hVM VMs

We assume that a Map-Reduce Job J_i consists of a set of map tasks M_i and a set of reduce tasks R_i, where $|M_i| \geq |R_i|$. The aggregate data size $\in J_i$ is be represented as $Size(J_i)$. It is also the total size of input datasets of the map tasks, hence $Size(J_i) == Size_{map}(J_i)$. On the other hand the total size of the input data of the reduce tasks is denoted by function $Size_{reduce}(J_i)$.

Furthermore, in this paper we consider three types of VMs which can host a map or a reduce task: $VM = \{SmallVM, MediumVM, LargeVM\}$, and the MIPS (millions instructions per seconds) rating of these VMs is denoted by set VM_{mips}. We assume that a *SmallVM* can only run one map task and one reduce task at a given point of time i.e., $SmallVM_{map} = 1$ and $SmallVM_{reduce} = 1$. On the other hand, $MediumVM_{map} = 2$, $MediumVM_{reduce} = 2$ and $LargeVM_{map} = 4$ and $LargeVM_{reduce} = 4$ respectively. Our modeling assumptions are based on Amazon EC2 [5] VM configurations where the number of processor core doubles across VM types. We also assume that each VM can

be allocated network bandwidth in proportion to their sizes. For example,the bandwidth allocation of each VM type is defined by the following relation: $4 * SmallVM_{bandwidth} = 2 * MediumVM_{bandwidth} = LargeVM_{bandwidth} = 4B(Mbps)$, where $VM_{bandwidth}$=$\{SmallVM_{bandwidth}, MediumVM_{bandwidth}, LargeVM_{bandwidth}\}$.

In the following, we first model the makespan of Map-Reduce Job followed by the Monetary cost model. Finally, we formalize the optimization problem in Section IV-C.

A. Makespan of Map-Reduce Job

As we discussed above, each type of VM has its corresponding capacity limit for processing the map and reduce tasks as shown in Eq. 1 and Eq. 2 respectively:

$$VM_{map} = \{SmallVM_{map}, MediumVM_{map}, LargeVM_{map}\} \quad (1)$$

$$VM_{reduce} = \{SmallVM_{reduce}, MediumVM_{reduce}, LargeVM_{reduce}\} \quad (2)$$

In general, Map-Reduce jobs have four stages [19]. In the first stage, input data is transferred to the Map-Reduce cluster. In the second stage, Map tasks process the data. In the third stage, shuffling of intermediate data is done and in last stage Reduce tasks aggregate the result set emitted by different Map tasks. Based on these four stages, we can model the makespan of a Map-Reduce Job by splitting it into four steps: *map data transfer*, *map task execution*, *reduce data transfer* and *reduces task execution*.

1) Network delay of transferring input data to Map task: Given a Map-Reduce Job J_i, we apply the Eq. 3 to calculate the network data transfer delay:

$$TT(M_i, vm_j) = \frac{\frac{Size_{map}(J_i)}{|M_i|}}{vm_j^{bw}},\ vm_j^{bw} \in VM_{bandwidth} \quad (3)$$

2) Map task execution delay: In order to calculate the execution time of map tasks, we model following equation:

$$MIMT(M_i) = mi_{map} * \frac{Size_{map}(J_i)}{|M_i|} \quad (4)$$

where mi_{map} is the millions of instructions per MB data when processing each map task. Therefore, the execution time of a map task on a given vm_j is:

$$TET(M_i, vm_j) = \frac{MIMT(M_i)}{vm_j^{mips}},\ vm_j^{mips} \in VM_{mips} \quad (5)$$

3) Network delay of transferring input data to Reduce task: As we have discussed above, the number of reduce tasks of J_i is always less than the number of map tasks. Moreover, the data size of the reduce tasks must also differ from that of map tasks. Thus, the network delay of transferring data to vm_j where a reduce task will be mapped is:

$$TT(R_i, vm_j) = \frac{\frac{Size_{reduce}(J_i)}{|R_i|}}{vm_j^{bw}},\ vm_j^{bw} \in VM_{bandwidth} \quad (6)$$

4) Reduce task execution delay: Similar to map task, we calculate the execution time of a reduce taks on a VM using the concept: MIRT (millions instructions of reduce task), which is defined as:

$$MIRT(R_i) = mi_{reduce} * \frac{Size_{reduce}(J_i)}{|R_i|} \quad (7)$$

where mi_{reduce} represents the millions of instructions per MB data when processing a reduce task. Equation 8 calculates the execution time of reduce task on vm_j:

$$TET(R_i, vm_j) = \frac{MIRT(R_i)}{vm_j^{mips}},\ vm_j^{mips} \in VM_{mips} \quad (8)$$

A Map-Reduce Job may not be executed in a single VM, the makespan of J_i over a set of map and reduce tasks mapped to VMs hVM can be computed as shown in Equation 9. The hVM has two subsets hVM' and hVM'', representing the VMs that executed map and reduce tasks respectively.

$$\begin{aligned} &makespan(J_i, hVM) = makespan_m(M_i, hVM') + \\ &makespan_r(M_i, hVM'') \\ &= \max_{vm_j \in hiredVM'} (TT(M_i, vm_j) + TET(M_i, vm_j)) \\ &+ \max_{vm_h \in hiredVM''} (TT(R_i, vm_h) + TET(R_i, vm_h)) \end{aligned} \quad (9)$$

, where $|M_i| = \sum_{vm_j \in hVM'} vm_j^{map}$ and, $|R_i| = \sum_{vm_h \in hVM''} vm_h^{reduce}$. vm_j^{map} is the maximum number of map tasks that vm_j can host, and vm_h^{reduce} is the maximum number of reduce tasks that vm_h can host.

B. Monetary Cost

In this paper, we assume that the VMs are charged under pay-as-you-go model (e.g. per minute). For example, the price of hiring or leasing computation time of a SmallVM for 30 minutes at Y dollar per minute base rate will be $30 * Y$ dollars. In our model, the MediumVM and LargeVM cost $2Y$ dollars and $4Y$ dollars per minute respectively. This is a reasonable and practical assumption and is modeled around the Amazon EC2 pricing scheme. In summary, the total cost of processing a Map-Reduce Job J_i on set of heterogeneous VMs will be:

$$\begin{aligned} COST(J_i, hVM) = (&makespan_m(M_i, hVM') * \\ &\sum_{vm_j \in hVM'} Y[vm_j \in SmallVM] + \\ &2Y[vm_j \in MediumVM] + \\ &4Y[vm_j \in LargeVM]) + \\ &(makespan_r(R_i, hVM'') * \\ &\sum_{vm_h \in hVM'} Y[vm_h \in SmallVM] + \\ &2Y[vm_h \in MediumVM] + \\ &4Y[vm_h \in LargeVM]) \end{aligned} \quad (10)$$

C. Optimization Problems

It is evident from Equation 4 and 7 that by increasing the level of parallelism for map and reduce tasks (i.e. hiring more and more number of VMs), the overall makespan can be

reduced. Furthermore, using more powerful VMs (Large vs. Small) has further potential to improve the makespan due to superior processor speed and network I/O capacity. However, adding more VMs and replacing small VM with larger VM will certainly lead to elevated monetary Cost. In other words, there are exsit a trade-off between Map-Reduce application makespan and monetary cost.

In this paper, we consider the following optimization problem *how to minimize the monetary cost to process a given set of Map-Reduce Jobs J while meeting the deadline C*. The problem is formalized as:

$$\begin{aligned} &\textbf{argmin} \sum_{J_i \in J} COST(J_i, hVM) \\ &\textbf{Subject to:} \sum_{J_i \in J} makespan(J_i, hVM) \leq C \end{aligned} \tag{11}$$

V. Proposed Map-Reduce Application Scheduling Algorithm (MASA)

In general, the scheduling problems (discussed above) are NP-hard problems as they map to 0-1 Knapsack problems [20]. Thus a heuristic approach is necessary to solve the problem. In this section, we give details of our proposed greedy algorithm which schedules Map-Reduce applications considering deadline and budget SLA requirements.

In Map-Reduce frameworks such as Hadoop, in general data is distributed across several cluster nodes where map tasks are scheduled in round robin fashion in order to have balanced load across each cluster node. In other words, each node will be executing a more or less equal number of map tasks considering homogeneous configuration across nodes. Thus, without loss of generality we can assume there is one large size map task running on each node instead of several small map tasks.

Algorithm 1 depicts the important steps of our algorithm. In the first step, the minimum and maximum number of map tasks are created such that user specified deadline and budget constraint can be achieved. These bounds are computed considering that the same type of VMs are used for executing tasks. In the next step, the algorithm iteratively computes (in a greedy fashion) the best possible number of mappers and reducers, and VM types that can minimize the cost, while meeting the deadline constraint.

VI. Evaluation

In this section, we evaluate the performance of our proposed MASA algorithm and compare it against existing Map-Reduce scheduling approach (NonSLA or SLA agnostic). We refer to the existing approach as "NonSLA Algorithm". In this algorithm, each user will specify which types of VM s/he intends to initiate to run his/her map-reduce tasks. The broker will calculate the maximum number of VMs required so that mapreduce job finished by end of the deadline. The broker then schedul the job after initiating these VMs.

Algorithm 1: MASA: Greedy based proposed algorithm

Data: Input: User Request = r1;// details of Map-Reduce application, deadline,budget
Result: Allocation aij;// allocation of map and Reduce tasks to VMs
Minmap=Calculate Lower bound on number of mappers;
Maxmap=Calculate Upper bound on number of mappers;
Let AllocList be the list of possible allocations; **for** $i \in (Minmap, Maxmap)$ **do**
  for $j \in (1, i)$ **do**
    // For allocation, choose VM types having minimum $\frac{cost(vmtype)}{VM^{cores} * VM^{MIPS}}$
    $a_{ij} = Compute\_Allocation(i, j, r1)$;
    // calculate time and cost of the above allocation
    $calculate\_time(r1, a_{ij})$;
    $calculate\_cost(r1, a_{ij})$; // compare the time with deadline
    if $time > deadline || cost > budget$ **then**
      delete this allocation;
    else
      if $time < deadline \& cost < budget$ **then**
        keep it as an option i.e. insert in AllocList
      end
      ;
    end
    // choose the minimum cost allocation
    $ChooseMinimum\_Alloc(AllocList)$;
    Deploy r1 based on chosen allocation;
  end
end

A. Experimental Setup

To model a real Public Cloud environment and map-reduce application scheduling scenario, we utilised IoTSIM [4]. Our simulation setup considers multiple Map-Reduce jobs with different deadlines being submitted to MR Cloud Brokers.

1) User Requests Generation: Typically, on the user's side, a request for deploying and executing Map-Reduce application consist of details of application characteristic along with SLA constraints, such as deadline, budget. Next, we discuss how SLA constraints are modeled in our experiments.

- **Deadline** is defined as the maximum time (upper bound) that user would like to wait until the Map-Reduce job finishes execution. The deadline is measured in minutes. Deadline is calculated based on the makespan. Let maxExTime represent the maximum makespan of a Map-Reduce application, Let minExTime represent the minimum makespan a Map-Reduce application, Then, the estimated execution time (α) = $\frac{maxExTime + minExTime}{2}$. Based on this, we derived three different classes of deadline as following:$tight\ \ deadline\ = 0.5 * \alpha$,

$medium\ deadline = \alpha$, and $relaxed\ deadline = 2 * \alpha$.

- **Budget** represents the money that each user is willing to pay for the execution of its Map-Reduce tasks. The budget is calculated is modeled as follows. Let maxCost represent the maximum cost required to process all tasks in Map-Reduce job. Let minCost represent the minimum cost required to process all tasks in a Map-Reduce job, Then

$$\beta = \frac{maxCost + minCost}{2}$$

Based on this, we derived three different classes of budgets that can be specified by a user as following: $low\ budget = 0.50 * \beta$, $medium\ budget = \beta$, and $high\ budget = 2 * \beta$.

- **Data Size is** the size of data that will be processed by the Map-Reduce job. The unit of data size is MB. For experiments, three types of Map-Reduce jobs are considered based on data size: short, medium and long. The medium job has 5 times more data size than the short job, while the long job has 10 times the data size of the short job. The data size of each Map-Reduce job is modeled based on a uniform distribution.
- **Size of Map Task** is defined by millions of instructions that need to be executed to process each KB (Kilobyte) data in the map phase. The unit of this parameter is Millions Instructions (MI) per KB. It is modeled using a uniform distribution.
- **Size of Reduce Task:** is defined by millions of instructions that need to be executed to process each KB data in the reduce phase. The unit of this parameter is Millions Instructions (MI) per KB. It is modeled using a uniform distribution.

2) VM Configuration Modeling: We consider three types of VMs: Small, Medium and Large. The VM configurations are listed in the Table VI-A2.

VM Type	Small	Medium	Large
number of CPUs in a VM	1	2	4
MIPS for each core	x	2x	4x
Cost per hour	y	2y	4y

3) Evaluation metrics: As the aim of MR Cloud Broker is to reduce the cost without missing the deadline and budget. The following two metrics are quantified during evaluation:

- **Average Makespan:** The average makespan shows how fast Map-Reduce jobs are executed. Let MK_i represent the make span of request i and is calculated using Equation 7. The N is the number of requests. Then, the average makespan is calculated by the following formulas:

$$average\ makespan = \sum_{i \in N} \frac{MK_i}{N}$$

- **Average Cost:** The average cost presents how much does it cost to process Map-Reduce jobs. Let TC_i represent the cost of processing request i and is calculated using Equation 8. Let N is the number of requests. Then, the average cost is calculated by the following formulas:

$$average\ cost = \sum_{i \in N} \frac{TC_i}{N}$$

4) Evaluation Scenarios: To understand the behavior and performance of our proposed MASA algorithm, We consider the following four types of scenarios; each varying different experimental parameters discussed above.

- Variation in the number of concurrent user requests.
- Variation in the VM configurations.
- Variation in the Deadlines
- Variation in Request Sizes.

B. Analysis of Results

1) Variation in Number of Concurrent User requests: In this scenario, we change the number of Map-Reduce Jobs (5, 10, 20) submitted simultaneously while keeping other independent variables (VM Configuration, cost model) at medium level. Each batch of Map-Reduce jobs consists of 30% short jobs, 30% medium jobs and 40% long jobs. Figure 2(a) and 2(b) clearly shows how our proposed algorithm MASA outperforms NonSLA based existing algorithm for executing Map-Reduce jobs on Public Clouds. MASA can achieve more than 50% lower cost and makespan.

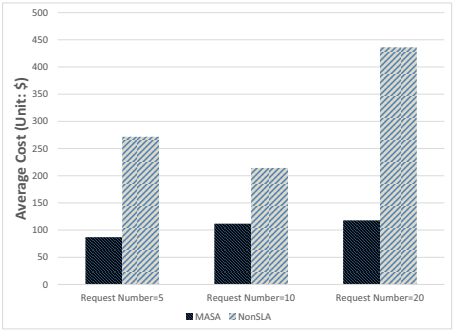
2) Variation in Deadline: In this scenario, we set the aforementioned different type of deadline (tight, medium or relaxed) for the same batch of Map-Reduce job requests.

Figure 3(a) and 3(b) show how the performance of MASA and NonSLA algorithms are affected by the deadline. It can be clearly seen that NonSLA approach performance is similar to MASA as deadlines become stricter. The principal reason behind this behaviour is that as deadlines get stricter, the set of possible VMs that can host map and reduce tasks is restricted and is often tilted towards superior VM configurations.

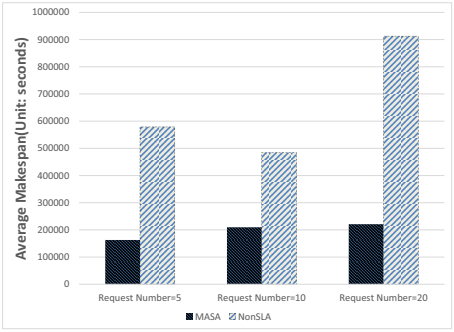
3) Variation in Request Sizes and Deadlines: In this scenario, we mix different types of deadline (tight, medium, relax) for different types of Map-Reduce jobs (short, medium, long) while keeping VM Configuration and cost model same. The deadline is varied as follows:

- **R4S,M4M,L4T** means we set relax deadline for short jobs, medium deadline for medium jobs and tight deadline for long jobs;
- **M4S,T4M,R4L** means we set medium deadline for short jobs, tight deadline for medium jobs and relax deadline for long jobs;
- **T4S,R4M,M4L** means we set tight deadline for short jobs, relax deadline for medium jobs and medium deadline for long jobs.

Figure 4(a) and 4(b) shows how MASA performs in comparison to NonSLA when different deadline distribution models are applied for different request mix. Overall MASA still is better at cost optimization in comparison to NonSLA algorithm. Even though overall makespan of job increases with variation in deadlines, MASA outperforms NonSLA counterpart.



(a) Average Cost



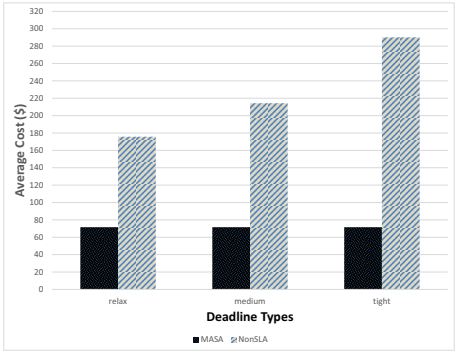
(b) Average Makespan

Fig. 2. Variation in Number of Jobs

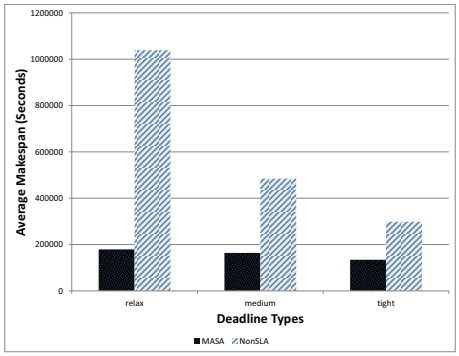
4) Variation in VM Configurations: In this scenario, we vary MIPS rating of VMs. We define three different types of MIPS Configurations (small, medium or high) as in the following:

- **Low MIPS** means the MIPS of medium VM is 1.5 times than that of small VM, and the MIPS of large VM is 1.5 times than that of medium VM;
- **Medium MIPS** means the MIPS of medium VM is 2 times than that of small VM, and the MIPS of large VM is 2 times than that of medium VM;
- **High MIPS** means the MIPS of medium VM is 2.5 times than that of small VM, and the MIPS of large VM is 2.5 times than that of medium VM.

Figure 4(a) and 4(b) shows how MASA performs in comparison to NonSLA. Overall MASA still incurs very low cost to the user in comparison to NonSLA algorithm. As difference between VMs's performance increases, both algorithms tend to select more number of large VMs than smaller VMs due to which the average makespan decreases considerably. However, as cost of Large VMs is much higher as compared against small VMs, the decrease in the average cost is only 25% from Low MIPS to High MIPS configuration.



(a) Average Cost



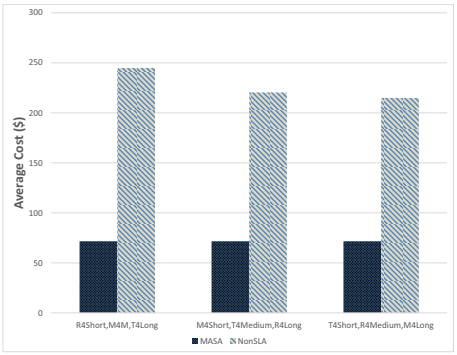
(b) Average Makespan

Fig. 3. Variation in Deadlines

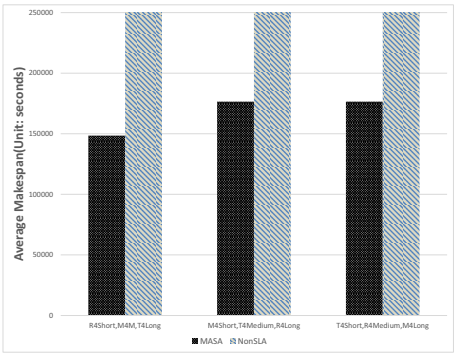
VII. CONCLUSIONS AND FUTURE DIRECTIONS

As BigData is gaining importance, more and more applications have been redesigned to use BigData frameworks such as Apache Hadoop that supports Map-Reduce programming model. These applications are generally hosted on Public Clouds which provide virtually infinite on-demand storage and computing resources. This paper identified an important gap in the literature concerning scheduling of Map-Reduce jobs on Public Clouds considering while SLA requirements of a user in terms of budget and deadline. To this end, we first modeled the scheduling problem and then proposed a novel scheduling algorithm MASA which: (i) computes in an greedy manner the best combination of VMs for scheduling Map and Reduce tasks and (ii) considers run-time uncertainties (e.g., availability, throughput, and utilization) during resource allocation process. MASA minimizes data analysis cost while avoiding SLA violations. The extensive IoTSim-based evaluation clearly shows that MASA can help users reduce cost of executing Map-Reduce applications on public clouds by about 25% to 60%. The cost saving efficiency of the proposed SLA-aware scheduling approach depends on the complexity (e.g., number of Mappers, number of Reducers, input data size, output data size) of Map-Reduce application.

In future, we will extend the current scheduling model to include other SLA constraints such as network bandwidth and storage. We will also extend our approach to take advantage of

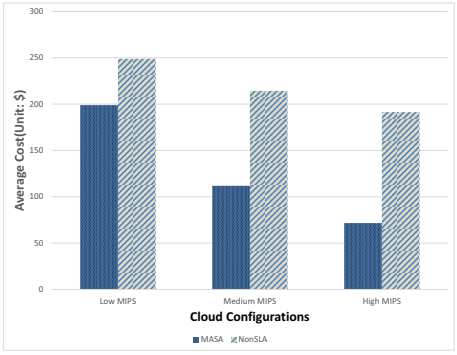


(a) Average Cost

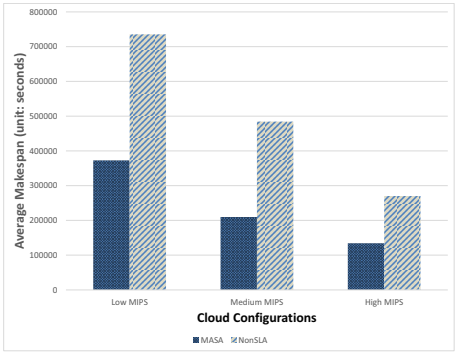


(b) Average Makespan

Fig. 4. Variation in Request Sizes and Deadlines



(a) Average Cost



(b) Average Makespan

Fig. 5. Variation in Cloud Configurations

Software Defined Networking (SDN) capabilities, specially as regards to optimizing data transfer delays between distributed file-system, Mappers, and Reducers. We will also investigate new scheduling approaches which can undertake joint optimisation of QoS parameters across VMs and SDN-enabled datacentre networking infrastructure.

REFERENCES

[1] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact." *MIS quarterly*, vol. 36, no. 4, pp. 1165–1188, 2012.

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

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. IEEE, 2010, pp. 1–10.

[4] X. Zeng, S. K. Garg, P. Strazdins, P. P. Jayaraman, D. Georgakopoulos, and R. Ranjan, "Iotsim: A simulator for analysing iot applications," *Journal of Systems Architecture*, 2016.

[5] J. Murty, *Programming amazon web services: S3, EC2, SQS, FPS, and SimpleDB*. " O'Reilly Media, Inc.", 2008.

[6] T. Redkar, T. Guidici, and T. Meister, *Windows Azure Platform*. Springer, 2011, vol. 1.

[7] "Cloud computing price comparison — cloudorado - find best cloud server from top cloud computing companies," http://www.cloudorado.com, accessed August 2016.

[8] G. Lee and R. H. Katz, "Heterogeneity-aware resource allocation and scheduling in the cloud." in *HotCloud*, 2011.

[9] K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing hadoop provisioning in the cloud." *HotCloud*, vol. 9, p. 12, 2009.

[10] H. Herodotou and S. Babu, "A what-if engine for cost-based mapreduce optimization." *IEEE Data Eng. Bull.*, vol. 36, no. 1, pp. 5–14, 2013.

[11] A. Verma, L. Cherkasova, and R. H. Campbell, "Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance," in *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2012, pp. 11–18.

[12] J. Polo, D. Carrera, Y. Becerra, M. Steinder, and I. Whalley, "Performance-driven task co-scheduling for mapreduce environments," in *2010 IEEE Network Operations and Management Symposium-NOMS 2010*. IEEE, 2010, pp. 373–380.

[13] K. Kc and K. Anyanwu, "Scheduling hadoop jobs to meet deadlines," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 388–392.

[14] Y. Wang and W. Shi, "Budget-driven scheduling algorithms for batches of mapreduce jobs in heterogeneous clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 306–319, 2014.

[15] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 495–504.

[16] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *2010 International Conference on Network and Service Management*. IEEE, 2010, pp. 9–16.

[17] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic distributed resource scaling for infrastructure-as-a-service," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, 2013, pp. 69–82.

[18] H. Yang, Z. Luan, W. Li, and D. Qian, "Mapreduce workload modeling with statistical approach," *Journal of grid computing*, vol. 10, no. 2, pp. 279–310, 2012.

[19] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[20] S. Martello and P. Toth, "An algorithm for the generalized assignment problem," *Operational research*, vol. 81, pp. 589–603, 1981.