Journal of Computational Science 26 (2018) 375–388

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Cost efficient scheduling of MapReduce applications on public clouds

Xuezhi Zeng[a,*], Saurabh Kumar Garg[b], Zhenyu Wen[c], Peter Strazdins[a], Albert Y. Zomaya[d], Rajiv Ranjan[e,f]

[a] Australian National University, Australia
[b] University of Tasmania, Australia
[c] School of Informatics, The University of Edinburgh, United Kingdom
[d] School of Information Technologies, University of Sydney, Australia
[e] Chinese University of Geoscience, Wuhan, China
[f] Newcastle University, United Kingdom

ARTICLE INFO

Article history:
Received 1 December 2016
Received in revised form 27 June 2017
Accepted 25 July 2017
Available online 9 August 2017

Keywords:
Big data
MapReduce
Cloud computing
Service level agreement
Scheduling
Cross layer

ABSTRACT

MapReduce framework has been one of the most prominent ways for efficient processing large amount of data requiring huge computational capacity. On-demand computing resources of Public Clouds have become a natural host for these MapReduce applications. However, the decision of what type and in what amount computing and storage resources should be rented is still a user's responsibility. 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 the 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 MapReduce applications. That is, they do not consider things such as number of map and reduce tasks that are needed to be scheduled and heterogeneity of Virtual Machines (VMs) available. This paper proposes a novel greedy-based MapReduce application scheduling algorithm (MASA) that considers the user's constraints in order to minimize cost of renting Cloud resources while considering Service Level Agreements (SLA) in terms of the user given budget and deadline constraints. The simulation results show that MASA can achieve 25–50% cost reduction in comparison to current SLA agnostic methods and there is only 10% performance disparity between MASA and an exhaustive search algorithm.

1. Introduction

The efficient processing of Big Data has become a predominant challenge in several emerging application domains including (but not limited to) enterprise computing, smart cities, remote healthcare, high energy physics, bio-informatics, and astronomy [1]. For example, online 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) [2–4]. With the push towards more automation for faster business strategy adaptation, most enterprises are moving towards the next generation Business Intelligence (BI) systems that can support data-driven decision making [5]. Such organizations often utilize MapReduce-based applications for efficient and effective large-scale processing of their Big Data. This requires either installation of a private Hadoop Cluster or deployment of MapReduce application on Public Cloud. Given the on-demand, vast and scalable computing and storage resources provided by Clouds, they are becoming more and more preferable deployment infrastructure.

Public Cloud providers such as Amazon Web Services have started to offer on-demand Hadoop clusters (PaaS), referred to as Elastic MapReduce, on its EC2 datacenters (IaaS) on pay-as-you-go basis [6]. However, current scheduling techniques and systems for deploying Hadoop clusters [7] on public IaaS Clouds are incapable of supporting Service Level Agreement (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 completing the data processing task. In the current practice, public Cloud providers require users (MapReduce application administrators) to manually decide the mix and type of IaaS resources they

* Corresponding author.
E-mail address: xuezhi.zeng@anu.edu.au (X. Zeng).

https://doi.org/10.1016/j.jocs.2017.07.017
1877-7503/© 2017 Elsevier B.V. All rights reserved.

need as part of their Hadoop cluster for finishing the analytic task over their Big Data within SLA constraints.

1.1. Research problem

Clearly, it is impossible to resolve such dependency between IaaS-level hardware configurations, deployment plan for Hadoop 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 Hadoop 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 (e.g., analyzing 100GB of Tweets in 10 min while subjecting 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 an NP-complete problem, and therefore intractable given current technology.

To be precise, we draw a simple example to demonstrate the challenge as NP-complete problem: We assume that a big data application contains 500 map jobs and 100 reduce jobs, and this application will be deployed in a Cloud cluster which has 100 available VMs. Therefore, there are $100^{(500+100)}$ possible allocations for deploying the given big data application.

The scheduling problem is further complicated by the fact that MapReduce 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 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 MapReduce 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.

1.2. Research methodology and contributions

The question of SLA-aware scheduling of applications has been addressed previously in the context of HPC, Grid, Cloud (at IaaS-layer), and database research over the last two decades. Our methodology differentiates itself in the 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 MapReduce application scheduling algorithm (MASA) that can pro-actively minimize the cost under user's constraints (budget and deadline), Big Data workload (data volume, priority) and IaaS performance (e.g., availability, throughput, and utilization) uncertainties. We extensively validate the performance of the SLA model and greedy MapReduce application scheduling algorithm (MASA) in the IoTSim simulator tool [8].

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 MapReduce application (aka. Jobs) on Public Clouds. Section 4 discusses our mathematical model and its assumptions. In Section 5, we present our proposed MapReduce application scheduling algorithms. Section 6 presents an evaluation of the performance of our proposed algorithms. In Section 7, we conclude the paper with future directions.

2. Related works

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 MapReduce application scheduling platforms (Apache YARN, Apache Mesos, Apache Spark, Amazon Elastic MapReduce) are designed for homogeneous clusters of hardware resources (VM, Storage, and Network). These platforms expect MapReduce 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 [9], Azure [10]) and academic projects (Cloudrado [11]) 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 [34,35]. Data and control flow dependencies in MapReduce applications are quite different from workflow as number of tasks in an application is 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.

On the other hand, most existing MapReduce research targets large-scale, clustered infrastructure environments [12,13]. In these works, authors proposed models for execution of MapReduce applications across multiple Clusters, and algorithms to minimize makespan of multiple MapReduce jobs on the same cluster. However, in this environment, each job is competing for the resources, which is not the case of a Public Cloud [14–18] and this type of MapReduce infrastructure is limited to task scheduling, ignoring resource selection and provisioning [13,19,20]. Wang et al. [21] 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 the context of the execution of MapReduce applications in public Clouds, most contributions focus on designing time efficient resource provisioning and task scheduling. Lee et al. [22] proposed dynamically allocating public Cloud resources to a Hadoop cluster based on a simple SLA constraint: minimize storage size. Kambatla et al. [23] suggested selecting the optimal set of public Cloud resources for Hadoop cluster by developing and profiling hardware resource consumption statistics. Similarly, the authors of [24] proposed 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. Mattess et al. proposed a policy for dynamic provisioning of Cloud resources to speed up execution of deadline-constrained MapReduce applications, by enabling concurrent execution of tasks, in order to meet a soft deadline for completion of the Map phase of the application. However, they only considered soft deadlines at the Map phase while ignoring reduce phase. In contrast in this paper, the execution of map and reduce tasks are scaled across a public Cloud, where resources can be virtually unlimitedly, with the goal of meeting user-defined deadline and budget. Moreover, we consider the data transmission time in our model which might significantly affect total makespan due to large data size.

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 trigger the scaling of cluster. There have been lots of research works conducted in developing performance models of MapReduce applications on Clouds. These performance models are based on machine learning [25], Markov and fast fourier transforms [26] or using wavelet techniques [27], and they provide predictions on execution (run) time of Map/Reduce tasks, input data volume, network I/O patterns, etc. Principal Component Analysis [28] has also been applied to learn performance model parameters of MapReduce applications. The progress made in MapReduce performance modeling is quite significant. However, these models cannot be implemented directly due to other constraints involved in making scheduling and deployment of MapReduce applications such as SLA constraints.

The preliminary results of this paper were reflected in our previous work HPCC 2016 [29]. The current work comprehensively extends earlier work. We proposed a new improved MapReduce application scheduling algorithm (MASA) and compared its performance with exhaustive search algorithm and NonSLA approaches. We have extensively expanded the experiments and thereafter drawn more conclusions.

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

3. System model: scheduling framework

Current Big Data processing platforms for deploying MapReduce applications can be divided into following layers:

- Platform as a Service (PaaS) level software frameworks such as Apache Hadoop [7] (an implementation of Google's MapReduce programming model [30]) that enables development of parallel data processing applications by exploiting capacity of a 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 scheduling, data staging, fault management, inter-process communication and result collection.
- Infrastructure as a Service (IaaS) that offers unlimited data storage and processing capabilities hosted in large datacenter farms.

Fig. 1 shows our scheduling scenario. In our approach, users submit their requests to a MR (MapReduce) Cloud broker whose responsibility is to select appropriate IaaS datacenter services for deploying Hadoop Cluster on behalf of a user (e.g., data analyst) while meeting user-specified SLA constraints. The users' deployment request consists 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 he/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.

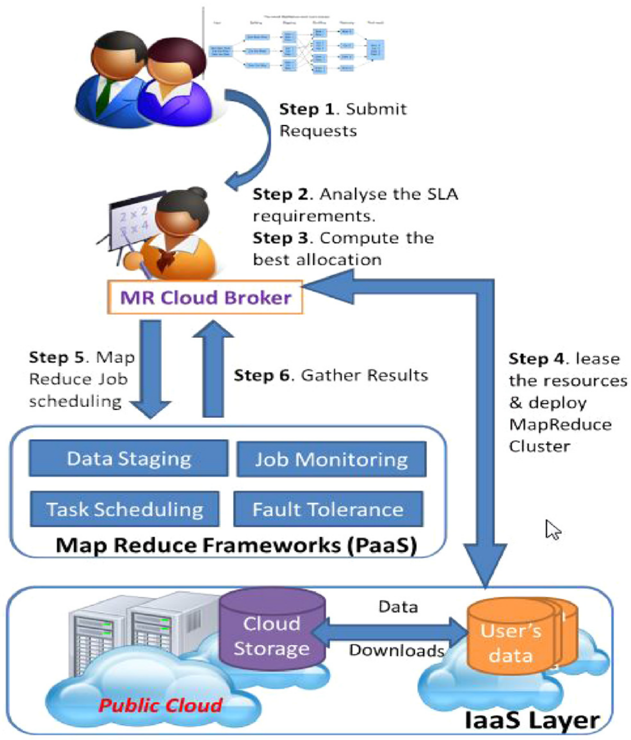

Step 1. Submit Requests
Step 2. Analyse the SLA requirements.
Step 3. Compute the best allocation
MR Cloud Broker
Step 5. Map Reduce Job scheduling
Step 6. Gather Results
Step 4. lease the resources & deploy MapReduce Cluster
Data Staging
Job Monitoring
Task Scheduling
Fault Tolerance
Map Reduce Frameworks (PaaS)
Cloud Storage
Data Downloads
User's data
Public Cloud
IaaS Layer

Fig. 1. MapReduce Job Scheduling on Public Cloud.

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 be executed. After analyzing the user's request, the MR broker deploys a MapReduce cluster after negotiating all the required IaaS services from the datacenter provider. In the next section, we will discuss the mathematical model and assumption that is considered as part of our scheduling approach.

4. Mathematical model and assumptions

MapReduce is a predominant framework for large-scale distributed data processing based on the divide and conquer paradigm. MapReduce works by breaking the processing into map and reduce phases. Map task and reduce task are executed in parallel on the different machines within the Hadoop cluster by MapReduce framework. Map performs filtering and sorting operations, and reduce performs summary operations. The user can specify map/reduce functions, and types of input/output. Applications that are processed using MapReduce programming framework (i.e., Hadoop) are called MapReduce applications.

4.1. MapReduce job model

The MapReduce job model consists of two phases: map phase and reduce phase as shown in Fig. 2. In the map phase, the application reads input data and generates intermediate keys. These computations are generally done in parallel by subtasks, which are called as map tasks. Then, in the reduce phase, another subtasks, called as reduce tasks, read intermediate keys and produce the program results.

Hence, we model a MapReduce Job J_i that consists of a set of map tasks M_i and a set of reduce tasks R_i, where $|\mathrm{M_i}| \geq |\mathrm{R_i}|$. The aggregate data size $\in J_i$ is be represented as $\mathrm{Size(J_i)}$. It is also the total size of

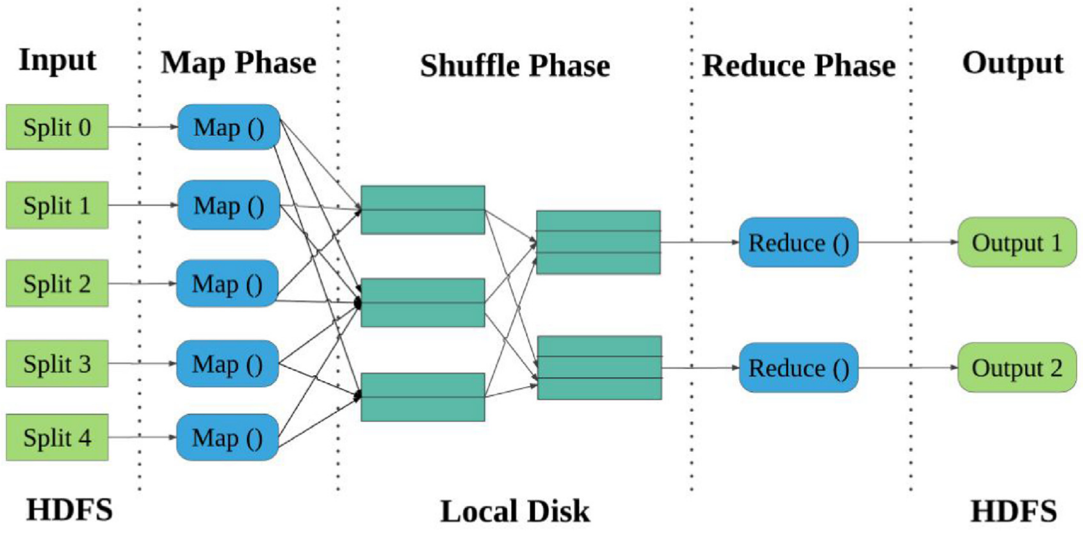


Fig. 2. The Structure of MapReduce Model.

input datasets of the map tasks, hence $\mathrm{Size}(J_i) \equiv \mathrm{Size}_{map}(J_i)$. On the other hand, the total size of the input data of the reduce tasks is denoted by function $\mathrm{Size}_{reduce}(J_i)$.

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 [9] 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: $4 \times SmallVM_{bandwidth} = 2 \times MediumVM_{bandwidth} = LargeVM_{bandwidth} = 4B(Mbps)$, where $VM_{bandwidth} = \{\mathrm{SmallVM}_{bandwidth}, \mathrm{MediumVM}_{bandwidth}, \mathrm{LargeVM}_{bandwidth}\}$.

For clarity and quick reference, we provide in Table 1 a summary of some symbols frequently used hereafter.

In the following, we first model the makespan of MapReduce Job followed by the monetary cost model. Finally, we formalize the optimization problem in Section 4.3.

4.2. Makespan of MapReduce job

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

$$VM_{map} = \{SmallVM_{map}, MediumVM_{map}, LargeVM_{map}\} \tag{1}$$

$$VM_{reduce} = \{SmallVM_{reduce}, MediumVM_{reduce}, LargeVM_{reduce}\} \tag{2}$$

In general, MapReduce jobs are processed in four stages [30]. In the first stage, input data is transferred to the MapReduce cluster. In the second stage, Map tasks process the data. In the third stage, shuffling of intermediate data is done and in the 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 MapReduce Job by splitting it into four steps: map data transfer, map task execution, reduce data transfer and reduces task execution. Fig. 3 denotes the composition of makespan by provisioning three VM resources (one Large VM, Medium VM, and Small VM respectively) and running a MapReduce job with three map tasks and two reduce tasks.

4.2.1. Network delay of transferring input data to map task

Given a MapReduce Job J_i, we apply Eq. (3) to calculate the network data transfer delay:

$$TT(M_i, VM_j) = C_0 + C_1 * \frac{\frac{Size_{map}(J_i)}{\mathrm{N_i}}}{VM_j^{bw}}, \quad VM_j^{bw} \in VM_{bandwidth}, \tag{3}$$

where C_0 and C_1 are constants.

4.2.2. Map task execution delay

In order to calculate the execution time of map tasks, we model following equation:

$$MIMT(M_i) = MI_{map} \times \frac{Size_{map}(J_i)}{|M_i|} \tag{4}$$

As denoted in Table 1, $\mathrm{MI_{map}}$ 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:

$$TET(Mi, VMj) = \frac{\mathrm{MIMT(M_i)}}{VM_j^{mips}}, \quad VM_j^{mips} \in VM_{mips} \tag{5}$$

4.2.3. 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 processed is:

$$TT(R_i, VM_j) = C_2 * N_i + C_3 * \frac{Size_{reduce}(J_i)}{VM_j^{bw}}, \quad VM_j^{bw} \in VM_{\mathrm{bandwidth}}, \tag{6}$$

where C_2 and C_3 are constants.

4.2.4. Reduce task execution delay

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

$$MIRT(M_i) = MI_{reduce} \times \frac{Size_{reduce}(J_i)}{|R_i|} \tag{7}$$

Table 1
Notation Frequently Used in Model and Algorithm Descriptions.

Symbol	Meaning
MapReduce Workload	
J	a MapReduce Job or workload set
J_i	a MapReduce 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 reduce tasks $\in J_i$
MI_{map}	the millions of instructions per MB data when processing each map task
MI_{reduce}	the millions of instructions per MB data when processing each reduce task
$Blocksize$	the default data block size that a distributed file system can store
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
$VM_{bandwidth}$	VM's network bandwidth
Y	the leasing cost of a small VM type for an hour
VM_j	a VM instance j
N_i	the number of VM that has been allocated for J_i
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
$Makespan_{map}(J_i,hVM')$	the makespan of executing map tasks of J_i over hVM' VMs
$Makespan_{reduce}(J_i,hVM'')$	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

As denoted in Table 1, MI_{reduce} is the millions of instructions per MB data when processing each reduce task. Therefore, the execution time of a reduce task on a given VM_j is:

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

A MapReduce 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 Eq. (9). The hVM has two subsets hVM' and hVM'', representing the VMs that executed map and reduce tasks respectively.

$$\begin{aligned} Makespan(J_i, hVM) &= Makespan_{map}(M_i, hVM') \\ &+ Makespan_{reduce}(R_i, hVM'') \\ &= \max_{VM_j \in hVM'}(TT(M_i, VM_j) + TET(Mi, VMj)) \\ &+ \max_{VM_h \in hVM''}(TT(R_i, VM_h) + TET(Ri, VMh)), \end{aligned} \tag{9}$$

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.

4.3. Monetary cost

In this paper, we have used the price schema of Amazon EC2 [31] to estimate the cost per hour of using a hosted VM. It is a reasonable and practical assumption since today many users rent public cloud resources (e.g. Amazon EC2) instead of maintaining their own private resources to run their MapReduce application.

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 min at Y dollar per minute base rate will be 30×Y dollars. Accordingly, in our model, the MediumVM and LargeVM cost 2Y dollars and 4Y dollars per minute respectively.

In summary, the total cost of processing a MapReduce Job J_i on set of heterogeneous VMs will be:

$$\begin{aligned} &COST(J_i, hVM) = \\ &Makespan_{map}(M_i, hVM') \times \\ &\sum_{VM_j \in hVM'} (Y \times [VM_j \in Small\ VM] + 2Y \times [VM_j \in MediumVM] \\ &+ 4Y \times [VM_j \in LargeVM]) + \\ &Makespan_{reduce}(R_i, hVM'') \times \\ &\sum_{VM_h \in hVM''} (Y \times [VM_j \in SmallVM] + 2Y \times [VM_j \in MediumVM] \\ &+ 4Y \times [VM_j \in LargeVM]) \end{aligned} \tag{10}$$

4.4. Optimization problems

It is evident from Eqs. (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

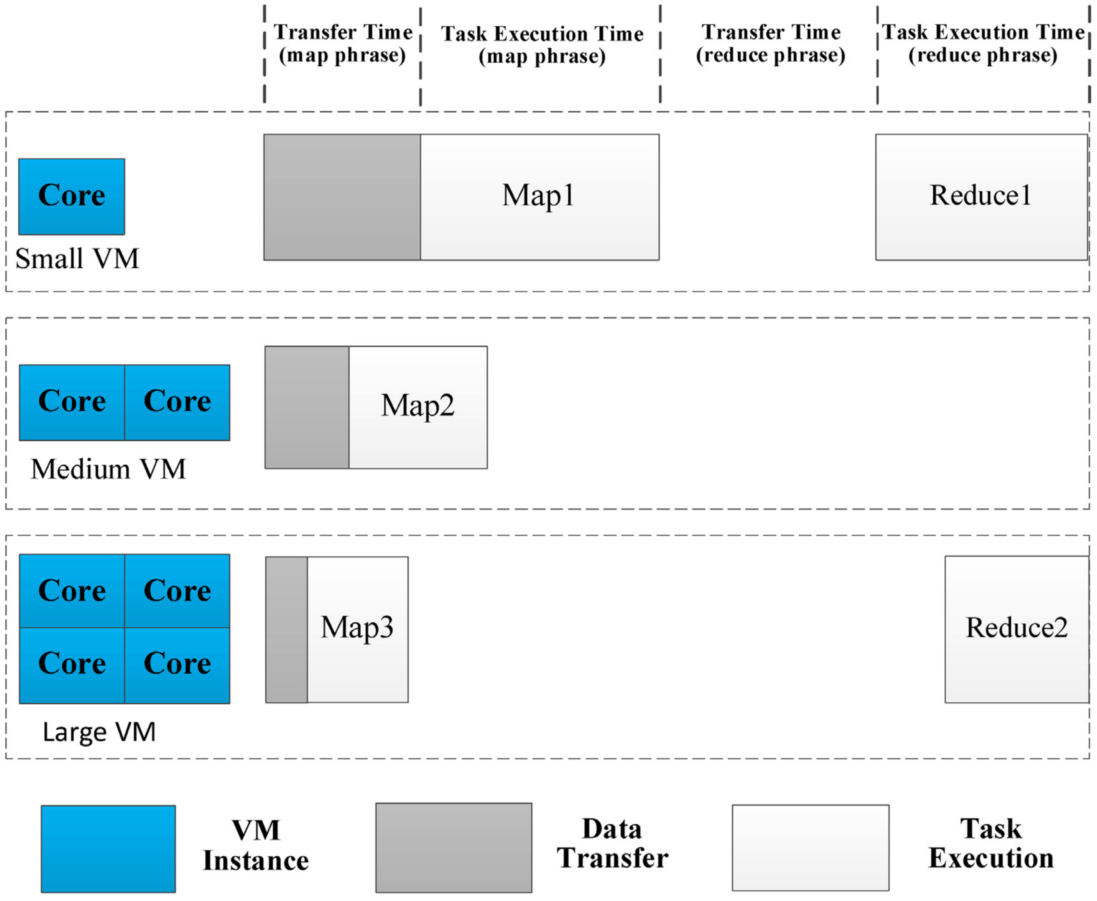

Transfer Time (map phrase)
Task Execution Time (map phrase)
Transfer Time (reduce phrase)
Task Execution Time (reduce phrase)
Core
Small VM
Map1
Reduce1
Core Core
Medium VM
Map2
Core Core Core Core
Large VM
Map3
Reduce2
VM Instance
Data Transfer
Task Execution

Fig. 3. The compositions of makespan when executing a MapReduce Job.

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 exists a trade-off between makespan and monetary cost for a MapReduce application.

In this paper, we consider the following optimization problem: How to minimize the monetary cost to process a given set of MapReduce Jobs J while meeting the deadline C.

The problem is formalized as:

$$\begin{aligned} &argmin \sum_{J_i \in J} \mathrm{COST}(J_i,hVM) \\ &Subjectto: \sum_{J_i \in J} \mathrm{Makespan}(J_i,hVM) \le C \end{aligned} \tag{11}$$

4.5. *Proposed MapReduce application scheduling algorithm (MASA)*

The provisioning and scheduling problem discussed above is a multidimensional knapsack problem that was shown to be NP-complete as they map to 0–1 Knapsack problems [32]. Thus a heuristic approach is necessary to solve the problem. In this section, we present the details of MapReduce application scheduling algorithm, called MASA with the goals of optimizing Eq. (11) considering the budget and deadline SLA requirements

In MapReduce 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 describes the significant steps of MASA algorithm. In the first step, the lower bound and upper bound number of map tasks are calculated such that user specified deadline and budget constraint can be achieved. For the sake of simplicity, the lower bound of map tasks equals one. The upper bound of map tasks can be calculated by $\frac{\mathrm{Size_{map}}(J_i)}{\mathrm{Blocksize}}$. As denoted in Table 1, $\mathrm{Size_{map}}(J_i)$ means the input data size of map tasks J_i. Blocksize represents the default data block size that a distributed file system can store. The second step is to decide the scope of reduce task number. Since the number of reduce task is smaller than the number of map task, we set the lower bound of reduce tasks as one and upper bound of reduce tasks as the number of map task. The main step followed is to compute allocation based on the pair of map tasks and reduce tasks for the coming request. Specifically, MASA iteratively computes (in a greedy fashion) all possible number of map tasks and reduce tasks according to the aforementioned lower bound and upper bound. Then, the number of same type VMs (SmallVM or Medium VM or LargeVM) is calculated (in Line 6). After that, MASA allocates each possible pair of map and reduce tasks into these VMs. For each potential allocation, makespan and cost are calculated according to the formulas discussed in above subsection. In each iteration, MASA discards the bad allocation that has either makespan exceeding deadline or cost going beyond budget. After the iteration (Line 4–19), a set of valid allocations is collected and stored in *AllocList*. Finally, we pick the allocation which has the minimal monetary cost from *AllocList* (see Line 21), and deploys the MapReduce jobs based the allocation.

Time complexity: we assume that N number of map jobs meets the Lower and Upper bound, and M number of available VMs, so the time complexity for allocating the map jobs in the worst case is O(N*M). Similarly, if the number of reduce jobs is N^*, therefore allocating the given reduce jobs in the worst case the time complexity is N^{**}M.

Algorithm 1: MASA

```
Data Input: User Request = r1; // details of MapReduce application and SLA
requirements(deadline and budget)

Result:   Allocation aij;     //allocation of map and reduce tasks to VMs

1.  Minmap = Calculate Lower bound on number of mappers;
2.  Maxmap = Calculate Upper bound on number of mappers;
3.  Let AllocList be the list of possible allocations;
4.  for i ∈ (Minmap, Maxmap) do
5.      for j ∈ (1,i) do
            //For allocation, calculate number for the same type of VM;
6.          CalculateNumberofVM;
7.          aij = Compute Allocation(i,j, r1);
8.          // calculate makespan and cost of the above allocation
9.          calculate makespan(r1,aij);
10.         calculate cost(r1, aij);     // compare the makespan with deadline
11.         if makespan > deadline || cost > budget then
12.             delete this allocation;
13.         else
14.             if makespan < deadline & cost < budget then
15.                 insert it into AllocList
16.             end;
17.         end;
18.     end;
19. end;
20.     // choose the minimum cost allocation from AllocList
21.     ChooseMimimum_Allocation(AllocList);
22.     Deploy r1 based on chosen allocation;
```

5. Evaluation

In this section, we evaluate the performance of our proposed MASA algorithm. In order to better demonstrate the performance of MASA, we use two algorithms as benchmark. The first one is existing MapReduce scheduling approach (NonSLA or SLA agnostic). We refer to the existing approach as NonSLA Algorithm. The second one is exhaustive search algorithm which gives a solution close to optimal. We comprehensively compare MASA against NonSLA algorithm and exhaustive search algorithm. The following sections give the details for NonSLA algorithm and exhaustive search algorithm respectively.

5.1. Benchmarking algorithms

5.1.1. NonSLA algorithm: existing scheduling approach

This algorithm describes existing MapReduce scheduling approach (NonSLA or SLA agnostic). In this algorithm, rather than specifying budget and deadline, each user will randomly specify which types of VM(Small VM, Medium VM or Large VM s/he intends to initiate to run his/her MapReduce tasks. The broker will calculate the maximum number of VMs required so that MapReduce job finished by the end of the deadline. The broker then schedules the job after initiating these VMs. As a consequence, NonSLA algorithm may meet the deadline, but the cost goes very high, or the cost may fall within the scope of the budget, but the deadline can't be met. Apparently, NonSLA algorithm doesn't target the trade-off between budget and deadline and is not able to find the optimized solution from both budget and deadline perspectives.

Algorithm 2: NonSLA Algorithm

```
Data Input: User Request = r1; // details of MapReduce application and deadline (d) and
budget(b) and VM type to initiate

Result:   Allocation aij;     //allocation of map and reduce tasks to VMs

          //Number of VMs to be initiated

1. NumofVMs = $\frac{b}{VMType^{cost} \times d}$;
2. aij = Compute_Allocation(NumofVMs,r1);
3. deploy request r1 based on aij;
```

5.1.2. *Exhaustive search algorithm*

Algorithm 3 depicts the procedures how exhaustive search algorithm works. Similarly, the lower bound and upper bound number of map tasks are calculated such that user specified deadline and budget constraint can be achieved. Following, in each iteration, the exhaustive search algorithm selects all possible VM set that consists of the aforementioned different type of VMs {SmallVM, MediumVM, LargeVM}. In the main step of computing allocation, the algorithm computes the best possible VM set upon all possible pair of number of map tasks and reduces tasks that can minimize the cost, while meeting the deadline constraint. The algorithm chooses the optimized allocation by searching all possible allocations; this is why we call it exhaustive search. Thus, it will give a solution that is very close to the optimal solution.

Algorithm 3: Exhaustive Search Algorithm

```
Data Input: User Request = r1; // details of MapReduce application and SLA
requirements(deadline and budget)

Result:     Allocation aij;          //allocation of map and reduce tasks to VMs

1.  Minmap  = Calculate Lower bound on number of mappers;
2.  Maxmap = Calculate Upper bound on number of mappers;
3.  Let AllocList be the list of possible allocations;
4.  for i ∈ (Minmap, Maxmap) do
5.      for j ∈ (1,i) do
```

5.2. *Experimental setup*

To model a real Public Cloud environment and MapReduce application scheduling scenario, we utilized IoTSIM [8]. Our simulation setup considers multiple MapReduce jobs with different deadlines being submitted to MR Cloud Brokers.

5.2.1. *User requests generation*

Typically, on the user's side, a request for deploying and executing MapReduce application consists of details of application characteristic along with SLA constraints, such as deadline and 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 MapReduce job finishes execution. The deadline is measured in minutes. Deadline is calculated based on the makespan. Let maxExTime represents the maximum makespan of a MapReduce application, Let minExTime represents the minimum makespan of a MapReduce application. Then, the estimated execution time $(\alpha) = \frac{\text{maxExTime}+\text{minExTime}}{2}$.

Based on this, we derived three different classes of deadline as follows:

$$tightdeadline = 0.5 \times \alpha$$

$$mediumdeadline = \alpha$$

$$relaxeddeadline = 1.5 \times \alpha$$

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

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

Based on this, we derived three different classes of budgets that can be specified by a user as follows:

$$lowbudget = 0.5 \times \beta$$

$$mediumbudget = \beta$$

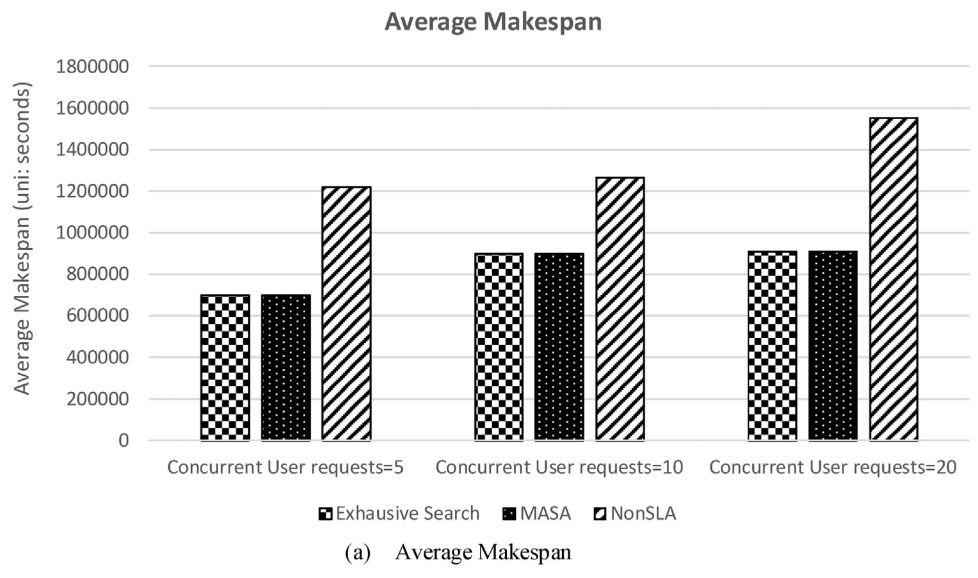
$$highbudget = 1.5 \times \beta$$

- Data Size is the size of data that will be processed by the MapReduce job. The unit of data size is MB. For experiments, three types of MapReduce jobs are considered based on data size: short, medium and long. The medium job has five times more data size than the short job, while the long job has ten times the data size of the short job. The data size of each MapReduce 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 MB (Megabyte) data when processing each map task during the map phase. The unit of this parameter is Millions Instructions (MI) per MB. It is modeled using a uniform distribution. It is worth to mention that each MapReduce application has different MI for map task depending on its characteristics of the job execution during map phrase. Verma et al. [33] extracted a single job profile using an automated profiling tool which can uniquely capture and demonstrate critical performance characterizes of the job execution for different MapReduce applications. It can be deduced that MI for map task represents the nature of different MapReduce applications.
- Size of Reduce Task, similarly is defined by millions of instructions that need to be executed to process each MB (Megabyte) data when processing each reduce task during the reduce phase. The unit of this parameter is Millions Instructions (MI) per MB. It is modeled as a uniform distribution. It is worth to mention that each MapReduce application has different MI for reduce task depending on its characteristics of the job execution during reduce phrase.

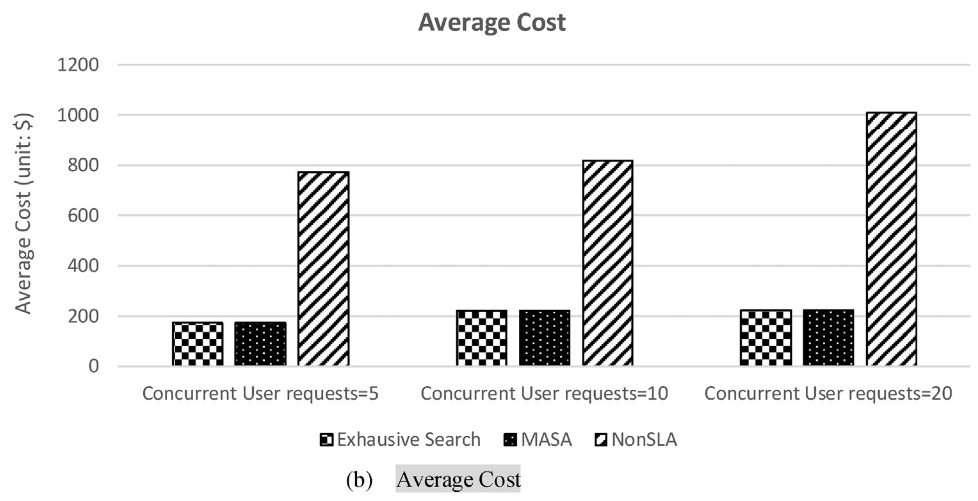
5.2.2. *Datacenter configuration*

Datacentre models the physical hardware that is offered by big data application provider. It encapsulates a set of computing hosts that can either be homogeneous or heterogeneous with their hardware configurations (memory, CPU, storage) where specific number of hosts and VMs are generated and run. Table 2 lists the datacentre configuration that is going to be consistent throughout the entire evaluation.

There are a few other parameters including OS, system architect to be defined for initialization. They are not factors that can affect



(a) Average Makespan



(b) Average Cost

Fig. 4. Variation in VM MIPS Configuration.

Table 2
Datacenter configuration.

pesNumber//number of cpus in a host	500
RAM//host memory (MB)	20480
Storage//host storage (MB)	1,000,000
Bandwidth//amount of bandwidth	1000
MIPS//millions of instructions per second	1000

our evaluation metrics. Hence, such parameters are irrelevant in our experiments.

5.2.3. *VM configuration modelling*

VM component stores the following characteristics related to a VM: processor, memory, storage size, bandwidth and MIPS. For a simplified reason, we define three types of VMs (Small, Medium, and Large) which is compatible with typical computing infrastructure in Amazon etc. provider. The configuration of three types of VMs is listed in Table 3.

5.2.4. *Evaluation metrics*

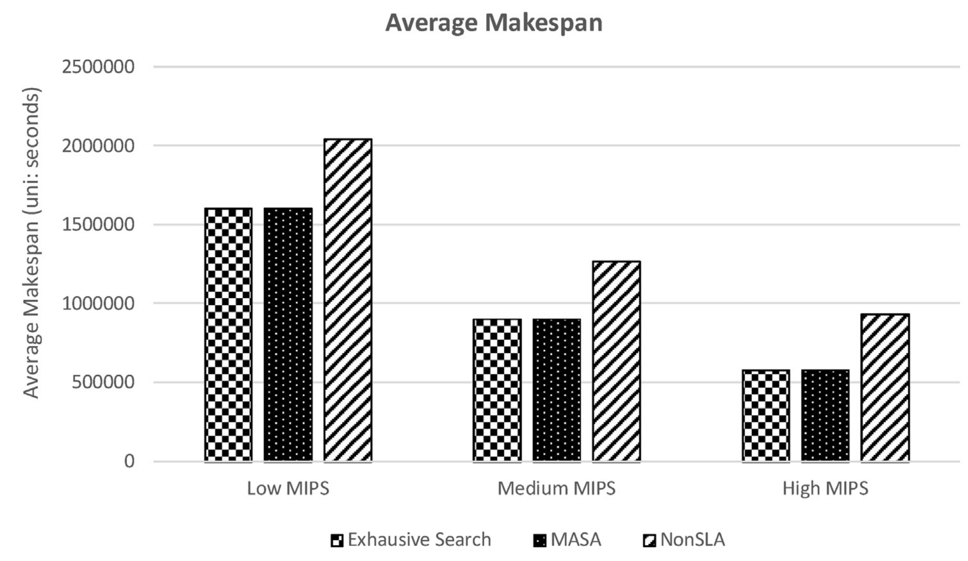
As the aim of MR Cloud Broker is to reduce the cost without missing the deadline and budget, the following two metrics are quantified during evaluation:

Table 3
VM configuration.

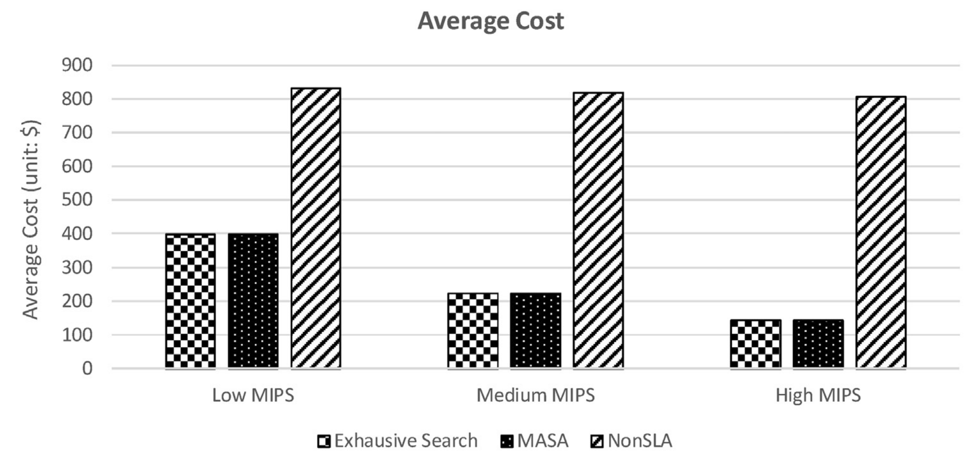
VM Type	Small	Medium	Large
Image Size//amount of storage (MB)	10,000	20,000	40,000
Ram//VM ram (MB)	512	1024	2048
MIPS for each core//millions of instructions per second	x	2x	4x
Bandwidth//amount of bandwidth	1000	1000	1000
pesNumber//number of CPUs in a VM	1	2	4
VMCostPerSec//the cost of processing in VM($)	1y	2y	4y

- Average Makespan: The average makespan shows how fast MapReduce jobs are executed. Let MK_i represent the makespan of MapReduce Job J_i and is calculated using Eq. (9). The N is the number of MapReduce jobs. Then, the average makespan is calculated by the following formulas:

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



(a) Average Makespan



(b) Average Cost

Fig. 5. Variation in Deadline.

- Average Cost: The average cost presents how much does it cost to process MapReduce jobs. Let TC_i represent the cost of processing MapReduce job J_i and is calculated using Eq. (10). Let N is the number of MapReduce jobs. Then, the average cost is calculated by the following formulas:

$$averagecost = \sum_{i \in N} \frac{TC_i}{N}$$

5.2.5. Evaluation scenarios

To understand the behavior and performance of our proposed algorithms, we consider the following four types of scenarios. In each scenario, different experimental parameters discussed above will be varied.

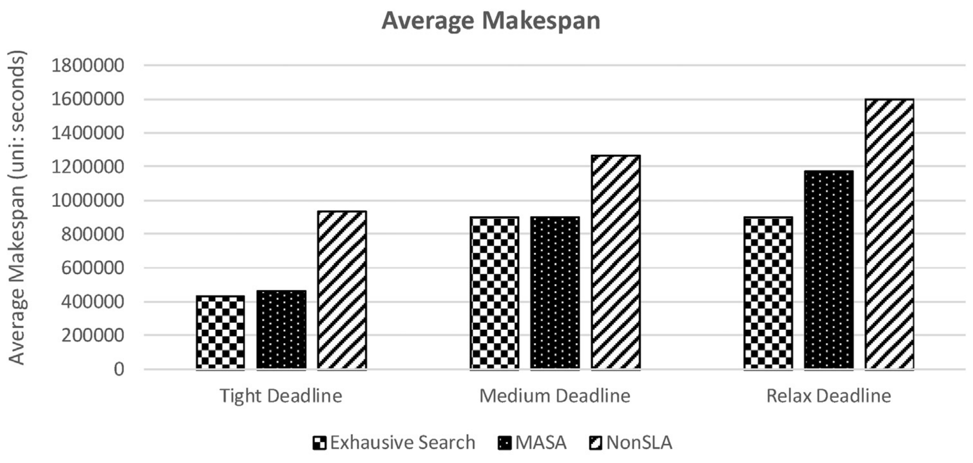
- Variation in the number of concurrent user requests
- Variation in the VM MIPS configurations
- Variation in the Deadlines
- Variation in the Budgets
- Variation in Request Sizes and deadlines

6. Analysis of experimental results

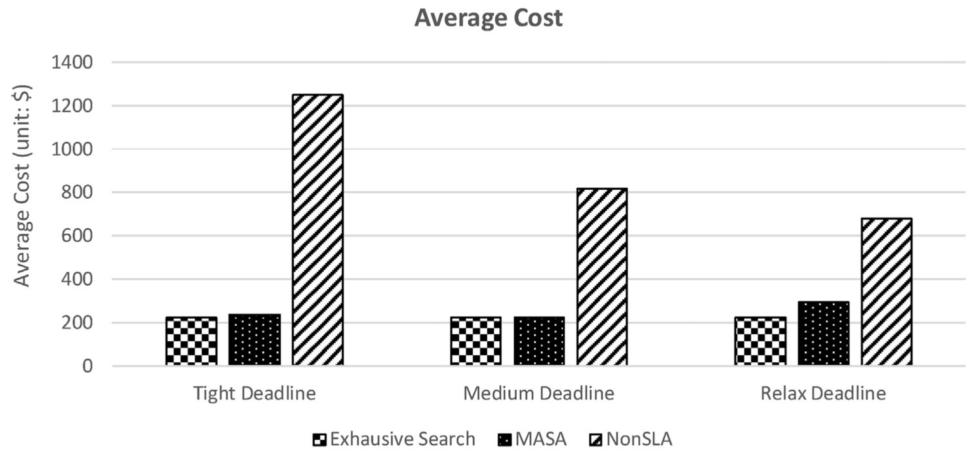
In this section, we analyze the experimental results upon the above evaluation scenarios and draw comprehensive conclusions about the performance of our proposed MASA algorithm against NonSLA and exhaustive search algorithm.

6.1. Variation in number of concurrent user requests

In this scenario, we change the number of MapReduce Jobs (5, 10, 20) submitted simultaneously while keeping other independent variables (VM Configuration, deadline and budget, etc.) at medium level. Each batch of MapReduce jobs consists of 30% short jobs, 30% medium jobs and 40% long jobs. Fig. 4(a) and (b) clearly shows how our proposed exhaustive search algorithm and MASA outperform NonSLA based existing algorithm for executing MapReduce jobs on Public Clouds. MASA can achieve very similar performance to the Exhaustive search algorithm and can achieve roughly 25% lower average makespan and 50% average costs.



(a) Average Makespan



(b) Average Cost

Fig. 6. Variation in Request Sizes and Deadlines.

6.2. *Variation in VM MIPS configuration*

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 two times that of small VM, and the MIPS of large VM is two times than that of medium VM;
- High MIPS means the MIPS of medium VM is 2.5 times that of small VM, and the MIPS of large VM is 2.5 times that of medium VM.

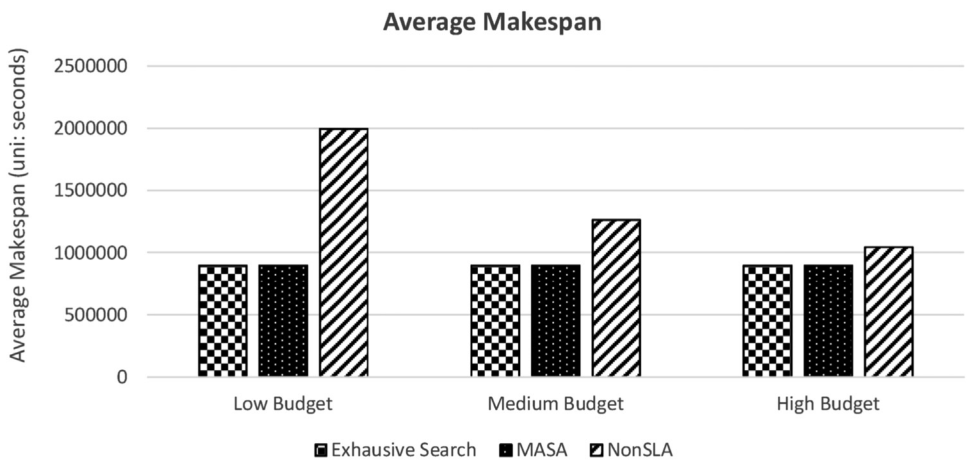
Fig. 5(a) and (b) shows how MASA performs in comparison to NonSLA algorithm and Exhaustive Search Algorithms. Overall MASA still incurs a very low cost to the user as compared to NonSLA algorithm and very close to the Exhaustive search algorithm. As difference between VMs's MIPS configuration increases, exhaustive search algorithm, and MASA 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 appropriately 25% from Low MIPS to High MIPS configuration.

6.3. *Variation in deadline*

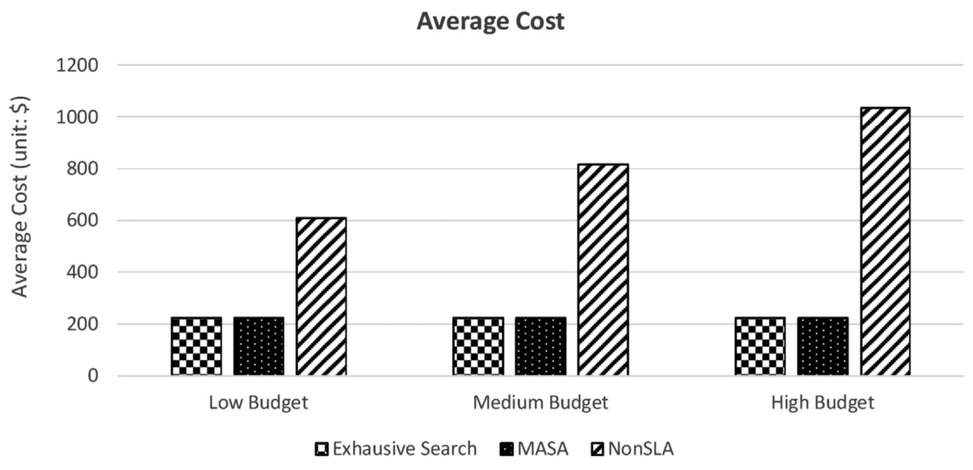
In this scenario, we set the aforementioned different type of deadlines (tight, medium or relaxed) for the same batch of MapReduce job requests. Fig. 6(a) and (b) shows how the performance of exhaustive search algorithm, MASA, and NonSLA algorithm are affected by the deadline. It can be clearly seen that NonSLA approach performance is more than 25% worse than the counterpart of exhaustive search algorithm and MASA in terms of average makespan as well as average cost. When deadline is relaxed, the exhaustive search algorithm can achieve just 10% lower average makespan and average cost than MASA. As deadline becomes stricter, MASA approach performance is similar to the exhaustive algorithm. The principal reason behind this behavior 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.

6.4. *Variation in budget*

In this scenario, we set the aforementioned different type of budget (low, medium or high) for the same batch of MapReduce job requests. Fig. 7(a) and (b) shows how different types of budget affect the performance of exhaustive search algorithm, MASA, and NonSLA algorithm. Still, NonSLA approach performance incurs higher average makespan and cost than the counterpart of exhaustive search algorithm and MASA. As budget become relaxed, the



(a) Average Makespan



(b) Average Cost

Fig. 7. Variation in Budget.

difference of average makespan between NonSLA and our proposed algorithm (MASA) becomes less. This is because that as budget get relaxed, more VMs that are often tilted towards superior VM configurations could be invested to achieve lower average makespan. It can also be observed, MASA still able to achieve an allocation very close to optimal.

6.5. Variation in request sizes and deadlines

In this scenario, we mix different types of deadline (tight, medium, relax) for different types of MapReduce jobs (short, medium, long) while keeping VM Configuration and cost model as the same. The deadline is varied as follows:

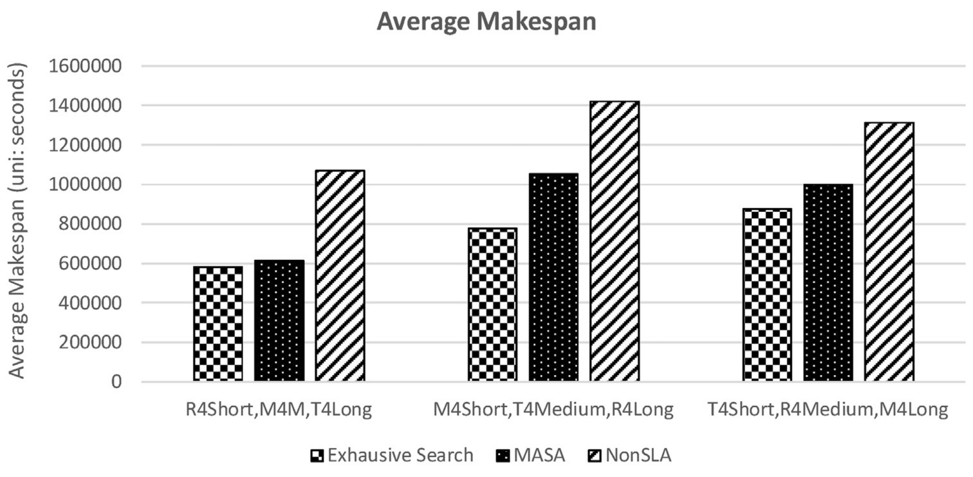
- R4S, M4M, L4T means we set relax deadline for short jobs, medium deadline for medium jobs and tight deadline for long jobs;
- M4S, T4M, R4L means we set medium deadline for short jobs, tight deadline for medium jobs and relax deadline for long jobs;
- T4S, R4M, M4L means we set tight deadline for short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

Fig. 8(a) and (b) shows how MASA performs in comparison to NonSLA when different deadline distribution models are applied for different mixed concurrent requests. Overall, MASA still is better at cost optimization comparing to NonSLA algorithm. Even though overall makespan of job increases, both exhaustive search algorithm and MASA outperform NonSLA counterpart. With variation in deadlines, MASA gives very close allocation to the exhaustive search and thus achieves a solution that cost within 10% of optimal average cost and average makespan.

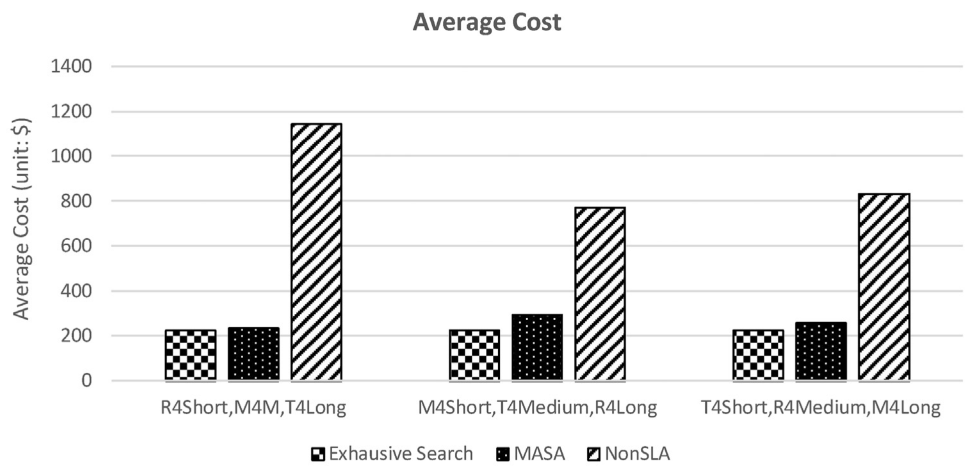
6.6. Discussion

Experimental results highlight the performance achieved through MASA while this paper does not consider the geo-distributed setting for optimizing the MapReduce applications. To this end, we first need to extend the models in IoTSim to correctly simulate the dynamic of the geo-distributed system. Further, a new algorithm which considers the influence of the available bandwidth is required. Moreover, the available bandwidth may change drastically due to the complex and unprediction of the internet.

In this paper, we do not consider that the interaction of our approach with Software Define Networking (SDN). The interaction can significantly benefit of optimizing data transfer delays between distributed file-system, Mappers, and Reducers. We will also investigate new scheduling approaches which can undertake joint optimization of QoS parameters across VMs and SDN-enabled datacenter networking infrastructure.



(a) Average Makespan



(b) Average Cost

Fig. 8. Variation in Request Sizes and Deadlines.

7. Conclusions

As Big Data is gaining importance, more and more applications have been redesigned to use Big Data frameworks such as Apache Hadoop that support MapReduce programming model. These applications are generally hosted on Public Clouds that provide virtually infinite on-demand storage and computing resources. This paper identified a significant gap in the literature concerning the scheduling of MapReduce jobs on Public Clouds considering while meeting SLA requirements of a user regarding budget and deadline.

To this end, we first modelled the scheduling problem and then proposed a novel MapReduce application scheduling algorithms (we call MASA) which: (i) computes greedily 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. Our proposed algorithm MASA minimize data analysis cost while avoiding SLA violations.

The extensive IoTSim-based evaluation clearly shows that our proposed algorithm MASA can help users reduce cost of executing MapReduce applications on public Clouds by about 25%–50%. 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 MapReduce application. Moreover, when we compared solution to optimal one, it achieves allocations which are very close to exhaustive search algorithm in most of the scenarios.

References

[1] L. Wang, W. Song, P. Liu, Link the remote sensing big data to the image features via wavelet transformation, Cluster Comput. 19 (2016) 793–810.
[2] L. Wang, J. Zhang, P. Liu, K.-K.R. Choo, F. Huang, Spectral–spatial multi-feature-based deep learning for hyperspectral remote sensing image classification, Soft Comput. (2016) 1–9.
[3] C. Yin, Z. Xiong, H. Chen, J. Wang, D. Cooper, B. David, A literature survey on smart cities, Sci. China Inf. Sci. 58 (2015) 1–18.
[4] L. Wang, K. Lu, P. Liu, R. Ranjan, L. Chen, IK-SVD: dictionary learning for spatial big data via incremental atom update, Comput. Sci. Eng. 16 (2014) 41–52.
[5] H. Chen, R.H. Chiang, V.C. Storey, Business intelligence and analytics: from big data to big impact, MIS Q. 36 (2012) 1165–1188.
[6] M.D. Assunção, R.N. Calheiros, S. Bianchi, M.a.S. Netto, R. Buyya, Big data computing and clouds: trends and future directions, J. Parallel Distrib. Comput. (2014).
[7] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (2010) 1–10.
[8] X. Zeng, S.K. Garg, P. Strazdins, P.P. Jayaraman, D. Georgakopoulos, R. Ranjan, IOTSim: a simulator for analyzing IoT applications, J. Syst. Archit. (2016).

[9] J. Murty, Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB, O'Reilly Media, Inc., 2008.

[10] Windows Azure: http://www.microsoft.com/windowsazure, 2012.

[11] Cloud computing price comparison — cloudorado - find best cloud server from top cloud computing companies, https://www.cloudorado.com/, Accessed August 2016.

[12] Y. Geng, S. Chen, Y. Wu, R. Wu, G. Yang, W. Zheng, Location-aware MapReduce in virtual cloud, 2011 International Conference on Parallel Processing (2011) 275–284.

[13] M. Mattess, R.N. Calheiros, R. Buyya, Scaling mapreduce applications across hybrid clouds to meet soft deadlines, 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA) (2013) 629–636.

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

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

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

[17] L. Wang, Y. Ma, J. Yan, V. Chang, A.Y. Zomaya, pipsCloud: high performance cloud computing for remote sensing big data management and processing, Future Generation Comput. Syst. (2016).

[18] L. Wang, D. Chen, Y. Hu, Y. Ma, J. Wang, Towards enabling cyberinfrastructure as a service in clouds, Comput. Electr. Eng. 39 (2013) 3–14.

[19] M. Alrokayan, A.V. Dastjerdi, R. Buyya, SLA-aware Provisioning and Scheduling of Cloud Resources for Big Data Analytics, 2014.

[20] G. Lee, N. Tolia, P. Ranganathan, R.H. Katz, Topology-aware resource allocation for data-intensive workloads, Proceedings of the First ACM Asia-pacific Workshop on Workshop on Systems (2010) 1–6.

[21] Y. Wang, W. Shi, Budget-driven scheduling algorithms for batches of MapReduce jobs in heterogeneous clouds, IEEE Trans. Cloud Comput. 2 (2014) 306–319.

[22] G. Lee, R.H. Katz, Heterogeneity-Aware resource allocation and scheduling in the cloud, HotCloud (2011).

[23] K. Kambatla, A. Pathak, H. Pucha, Towards optimizing hadoop provisioning in the cloud, HotCloud vol. 9 (2009) (p. 12).

[24] H. Herodotou, S. Babu, A what-if engine for cost-based MapReduce optimization, IEEE Data Eng. Bull. 36 (2013) 5–14.

[25] A. Matsunaga, J.A. Fortes, On the use of machine learning to predict the time and resources consumed by applications, Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (2010) 495–504.

[26] Z. Gong, X. Gu, J. Wilkes, Press: predictive elastic resource scaling for cloud systems, 2010 International Conference on Network and Service Management (2010) 9–16.

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

[28] H. Yang, Z. Luan, W. Li, D. Qian, MapReduce workload modeling with the statistical approach, J. Grid Comput. 10 (2012) 279–310.

[29] X. Zeng, S.K. Garg, Z. Wen, P. Strazdins, L. Wang, R. Ranjan, SLA-aware scheduling of map-Reduce applications on public clouds, The 18th IEEE International Conference on High Performance Computing and Communications (HPCC 2016) [Bibtex] (2016).

[30] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (2008) 1–13.

[31] C. Vecchiola, X. Chu, M. Mattess, R. Buyya, Aneka – Integration of Private and Public Clouds, Cloud Computing: Principles and Paradigms, Wiley Press, New York, USA, 2011, pp. 249–274, February.

[32] S. Martello, P. Toth, An algorithm for the generalized assignment problem, Oper. Res. 81 (1981) 589–603.

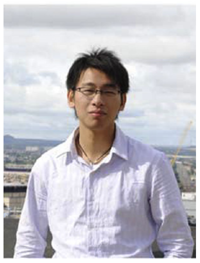
[33] A. Verma, L. Cherkasova, R.H. Campbell, Resource provisioning framework for mapreduce jobs with performance goals, ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing (2011) 165–186.

[34] Wen Zhenyu, Cala Jacek, Watson Paul, Romanovsky Alexanderl, Cost effective, reliable and secure workflow deployment over federated clouds, IEEE Transactions on Services Computing (2016).

[35] Wen Zhenyu, Qasha Rawaa, Zequn Li, Ranjan Rajiv, Watson Paul, Romanovsky Alexander, Dynamically partitioning workflow over federated clouds for optimising the monetary cost and handling run-Time failures, IEEE Transactions on Cloud Computing (2016).

Xuezhi Zeng is a PhD Candidate in the Computer Systems Group of the Research School of Computer Science at the Australian National University. He received his master degree information system from Fudan University. His research interests include distributed computing technologies, big data analytics, and internet of things.

Saurabh Garg is currently working as a lecturer in the Department of Computing and Information Systems at the University of Tasmania, Hobart, Tasmania. He was one of the few Ph.D. students who completed in less than three years from the University of Melbourne in 2010. He has published more than 30 papers in highly cited journals and conferences with Hindex 20. His doctoral thesis focused on devising novel and innovative market-oriented metascheduling mechanisms for distributed systems under conditions of concurrent and conflicting resource demand. He has gained about three years of experience in the Industrial Research while working at IBM Research Australia and India.



Zhenyu Wen received the M.S and Ph.D. degree in computer science from Newcastle University, Newcastle Upon Tyne, U.K., in 2011 and 2015. He is currently a PostDoctoral Researcher with the School of Informatics, the University of Edinburgh, Edinburgh, U.K. He has authored a number of research papers in the field of cloud computing. His current research interests include Multi-objects optimisation, Crowd sources, Artificial Intelligent and Cloud Computing.

Peter Strazdins is an Associate Professor in the Computer Systems Group of the Research School of Computer Science at the Australian National University. He is a Senior Member of the IEEE and a Senior Fellow of the Higher Education Academy (SFHEA). Peter is the convenor for the Bachelor of Advanced Computing. He is also a Green Rep, a Delegate to the National Tertiary Education Union. From 2009–2013, he was the Associate Director of Education for RSCS ANU. Up to 2009, he was Convenor of the Coursework Masters programs. the CSIT Safety Co-ordinator, the chair of the CSIT Occupational Health and Safety Committee and co-ordinator of the CSIT Ride to Work Group.

Albert Y. Zomaya is currently the Chair Professor of High Performance Computing & Networking and Australian Research Council Professorial Fellow in the School of Information Technologies, The University of Sydney. He is also the Director of the Centre for Distributed and High Performance Computing which was established in late 2009. Professor Zomaya is the author/co-author of seven books, more than 450 publications in technical journals and conferences, and the editor of 14 books and 19 conference volumes. He is the Editor in Chief of the IEEE Transactions on Computers and Springer's Scalable Computing Journal serves as an associate editor for another 19 journals including some of the leading journals in the field.

Dr. Ranjan has been able to establish an international reputation as a leader in the field of Scalable Computing and in particular Cloud Computing and Big Data Analytics. For over 10 years, he has conducted seminal research around the development of generic resource management models and methods for efficient scheduling and resource allocation for all types of parallel and distributed computing systems such as grid computing and cloud computing. He also easily extended his methods to new computing trends we see in the industry such as Internet of Things (IoT) and Edge Computing. As a prolific researcher and as of November 2016, he has published 180 scientific publications that include 113 journal articles, 47 conference papers, 12 book chapters, and 8 edited research books. His research has received excellent recognition from the research community as evidenced by the citations, Google scholar h-index 33, i-10 index 66, 6700+ citations (November 2016).