SLA-Based Resource Scheduling for Big Data Analytics as a Service in Cloud Computing Environments

Yali Zhao, Rodrigo N. Calheiros, Graeme Gange, Kotagiri Ramamohanarao, Rajkumar Buyya
Department of Computing and Information Systems
The University of Melbourne, Australia
{yaliz1@student., rnc@, gkgange@, kotagiri@, rbuyya@}unimelb.edu.au

***Abstract*— Data analytics plays a significant role in gaining insight of big data that can benefit in decision making and problem solving for various application domains such as science, engineering, and commerce. Cloud computing is a suitable platform for Big Data Analytic Applications (BDAAs) that can greatly reduce application cost by elastically provisioning resources based on user requirements and in a pay as you go model. BDAAs are typically catered for specific domains and are usually expensive. Moreover, it is difficult to provision resources for BDAAs with fluctuating resource requirements and reduce the resource cost. As a result, BDAAs are mostly used by large enterprises. Therefore, it is necessary to have a general Analytics as a Service (AaaS) platform that can provision BDAAs to users in various domains as consumable services in an easy to use way and at lower price. To support the AaaS platform, our research focuses on efficiently scheduling Cloud resources for BDAAs to satisfy Quality of Service (QoS) requirements of budget and deadline for data analytic requests and maximize profit for the AaaS platform. We propose an admission control and resource scheduling algorithm, which not only satisfies QoS requirements of requests as guaranteed in Service Level Agreements (SLAs), but also increases the profit for AaaS providers by offering a cost-effective resource scheduling solution. We propose the architecture and models for the AaaS platform and conduct experiments to evaluate the proposed algorithm. Results show the efficiency of the algorithm in SLA guarantee, profit enhancement, and cost saving.**

I. INTRODUCTION

As we enter into the big data era, data analytics becomes critical in decision making and problem solving for various domains, such as science, engineering, commerce, and industry. Big data is referred to a massive volume of structured, unstructured, semi-structured, or real-time data, which is difficult to manage and process with traditional database techniques [1, 2]. Huge amount of data is generated from various sources, i.e., social media, sensors, websites, and mobile devices every day. The key enablers of big data are the increased capability of storage and computing resources enabled by recent advances in Cloud computing and the increased availability of data and the ease of access to data supported by various companies like Amazon, Google, IBM, and Facebook. Cloud computing is a suitable platform for big data analytics by providing various resources, which are: (1) Cloud infrastructures, such as Amazon EC2, which offer on-demand virtualized resources, i.e., Virtual Machines (VMs) and storages; (2) Cloud platforms, such as Google App Engine, which manage underlying infrastructure resources and allow users to deploy and run applications; (3) application services, such as IBM Social Media Analytics [3], which are provided to users through web browser portals.

Analyzing big data to find potential insights in the data is essential for individuals, organizations, and governments to make better decisions, i.e., product trend prediction, business strategy making, and disaster prediction and management. However, big data analytics has several challenges. Obtaining analytic solutions is usually expensive as it requires large data storage systems to store voluminous data and considerable computing resources to analyze the data. Furthermore, licenses of Big Data Analytic Application (BDAA) are often expensive. As a result, big data analytics is mostly used by a small number of large enterprises [4]. Moreover, scheduling resources for BDAAs requires expert knowledge. For example, in order to reduce the time of generating analytic solutions, BDAAs should be deployed on various machines for parallel processing on large datasets, while in order to save energy and cost, resources should be scaled up and down for varied resource demands of BDAAs. The lack of such expertise makes resource scheduling for BDAAs difficult for many users. The above challenges illustrate the need for an Analytics as a Service (AaaS) platform, which aims at serving on-demand analytic requests and delivering AaaS to users as consumable services in an easy to use way and at lower cost. The delivered AaaS should satisfy Quality of Service (QoS) requirements of requests with Service Level Agreement (SLA) guarantees. SLA is the agreement negotiated between service users and providers, which defines the metrics, expected QoS, and penalties during service delivery.

To support the AaaS platform, our research focuses on efficiently scheduling Cloud resources for BDAAs and provisioning BDAAs to users as consumable services. We aim to serve the interests of AaaS providers to enhance profit by proposing cost-effective resource scheduling solutions, increase market share by accepting more requests, and improve reputation by delivering satisfactory services to users with SLA guarantees. BDAAs that typically process read-only query requests on given datasets and leave out data consistency and security issues [5] are our targeted applications. The key **contribution** of our work is an admission control and resource scheduling algorithm that admits queries based on QoS requirements, delivers analytic solutions to users with SLA guarantees, and provides a cost-effective resource scheduling solution to create higher profit for AaaS providers. To achieve this, we (1) propose the architecture and models of the AaaS platform; (2) formulate the resource scheduling problem based on mixed Integer

Linear Programming (ILP) model [6] and propose the admission control and resource scheduling algorithm; and (3) conduct experiments to evaluate the performance of the algorithm in SLA guarantee, profit enhancement, and cost saving.

II. Architecture and Models

A. Architecture

We propose the architecture of the AaaS platform that provisions general AaaS to various domains of users as consumable services, as shown in Fig. 1. The architecture is composed of the following components:

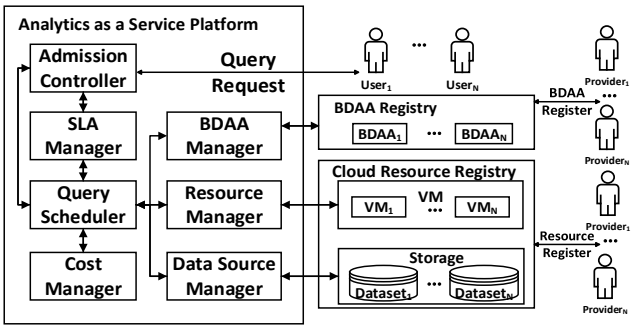

Figure 1. Architecture of the AaaS Platform

Admission controller makes decisions about whether to accept queries submitted by users. Admission controller first searches exhaustively in BDAA registry for requested BDAAs and the Cloud resource registry for all possible resource configurations. Then, it estimates query execution time and cost as basis to decide whether to accept or reject queries. Accepted queries are submitted to SLA manager.

SLA manager builds SLAs for accepted queries. SLA violations not only decrease user satisfaction but also generate additional penalty cost; thus, they should be avoided.

Query scheduler is the core component of the AaaS platform, which makes scheduling decisions and coordinates the other components. Scheduling decisions mainly contain: (a) **resource configuration**: the scheduler decides which type of Cloud resource to use and how many to use for each resource type; (b) **VM control**: the scheduler sends VM controlling commands to the resource manager to control VMs, i.e., create VM, terminate VM, and migrate VM; (c) **BDAA selection**: the scheduler selects requested BDAA to execute a query; (d) **query execution sequence**: queries that request the same BDAA are submitted to the waiting queues of VMs running the BDAA (as queries contain deadline constraints, they are ordered for execution to avoid deadline violations); and (e) **query management**: the scheduler monitors and manages status of queries during their lifecycles. Query status can be one of submitted, accepted, rejected, waiting for execution, being executed, succeeded, and failed. Query status is monitored to allow the scheduler to take appropriate actions, i.e., send query with failed status to cost manager to generate penalty cost.

Cost manager manages all the cost occurred in the AaaS platform, i.e., query cost and resource cost. It also aims at providing pricing policies that can attract more users to enlarge market share and generate higher profit.

BDAA manager manages BDAAs provided by different BDAA providers and keeps up-to-date BDAA information.

Data source manager manages datasets that are to be processed. As big data has high volume, we move the compute to the data to save data transferring time and network cost.

Resource manager keeps a catalog of all available Cloud resources from different providers and monitors status of VMs to allow the scheduler to take actions such as terminating idle VMs at the end of billing period to save cost.

B. Models

The models of the AaaS platform are shown as below:

Query request model: Query requests should contain the following information in query specifications: (a) QoS requirements, which contain budget and deadline to fulfill; (b) required resources to execute a query; (c) the requested BDAA; (d) data characteristics including data size, data type, and data location; (e) user who submits the query; and (f) query type, i.e., aggregation query and join query.

Cloud resource model contains a set of datacenters and a matrix showing the network bandwidth between the datacenters. Each datacenter contains a set of hosts and data storages that pre-store datasets. Each host contains a set of VMs.

BDAA profile model contains resource configurations, data processing time, data size, data type, and costs for different queries. BDAA profiles are the basis for AaaS platform to estimate query time and cost to make admission and scheduling decisions. Obtaining the profiles requires expert knowledge. Thus, BDAA profiles are assumed to be provisioned by BDAA providers and are reliable.

Cost model contains sub-models of resource cost, penalty cost of SLA violations, query cost (query income) that charges users for using AaaS platform, and profit that is created for AaaS platform providers. Profit is the difference of query cost and the sum of resource cost and penalty cost. Query cost has different cost policies, which are: (a) cost based on urgency of deadline; (b) proportional cost, i.e., proportional to BDAA cost; and (c) a combination of (a) and (b). Resource cost contains BDAA cost and Cloud resource cost. BDAA cost policies are: (a) fixed cost, i.e., annual contract; (b) usage period based cost, i.e., hourly basis; and (c) per request based cost. Penalty cost policies are: (a) fixed cost; (b) delay dependent cost; and (c) proportional cost [7].

III. Admission Control and Resource Scheduling

We propose the admission control and resource scheduling algorithm that only admits queries whose QoS can be satisfied and schedules resources to execute queries with SLA guarantees to maximize the profit of AaaS providers. We adopt proportional query cost model, which calculates the query cost proportional to BDAA cost and fixed BDAA cost model, which is fixed cost under annual contract. We aim to make scheduling decision that can give 100% SLA guarantee. Thus, penalty cost of violating SLAs can be avoided if scheduling decision is well made. Since query cost and BDAA cost are fixed and penalty cost can be avoided, our goal of

maximizing profit of the AaaS providers is to minimize the Cloud resource cost.

A. Admission Control Algorithm

The admission controller makes query admission decisions. It first searches the BDAA registry to check whether a query requested BDAA exists. If the BDAA exists, admission controller obtains the BDAA profile. Based on the profile, expected execution time and query cost for different resource configurations are calculated. The expected query cost is calculated based on the adopted query cost model. The expected finish time is the sum of estimated execution time, specified timeout (the maximum time the scheduling algorithm can run), VM creation time (the time to create new VM when needed), submission time (the time when the query is submitted), and waiting time (the time till the query is scheduled). Admission controller makes query acceptance decision if estimated finish time and cost of queries can satisfy QoS requirements of queries using any resource configuration. Afterwards, SLA manager builds SLAs for accepted queries.

B. Scheduling Algorithms

After the admission controller accepts queries, the query scheduler makes the scheduling decision for each BDAA. We develop scheduling algorithms that support different scheduling scenarios, which are periodic scheduling that schedules queries for each Scheduling Interval (SI) and non-periodic (real time) scheduling that schedules queries whenever they arrive. For periodic scheduling, we set different SI parameters to study how SI can influence the performance of the scheduling algorithms. We detail the study of algorithm performance for different scheduling scenarios in performance evaluation section.

We aim at scheduling and provisioning resources for BDAAs that can maximize profit for AaaS providers by minimizing resource cost and guarantee SLAs of queries on deadline and budget. The scheduling algorithms adopt a two-phase scheduling policy that provision resources in a scalable and elastic way. They scale resources down by releasing resources when the provisioned resource capacity is more than required to reduce cost and scales resources up by leasing new resources when provisioned resources do not have sufficient capacity to execute queries to avoid SLA violations.

1) ILP scheduling algorithm

We model and formulate the two-phase resource scheduling problem based on ILP model [6]. Phase 1 aims at minimizing resource cost by scheduling queries to existing VMs to maximize VM utilization while satisfying SLAs of queries. If queries are not successfully scheduled, they will be handled by Phase 2 for further scheduling, which aims to minimize the cost of creating VMs to execute queries with SLA guarantees.

Phase 1 of the scheduling problem is formulated as a multi-objective ILP problem [8]. Minimizing cost of resources for query execution under budget and deadline constrains is an NP-complete decision problem that can be polynomially reduced to an ILP problem. The optimization objective of Phase 1 scheduling is minimizing resource cost. To achieve this, we have three individual objectives to fulfill, which are: **Objective A** aims to maximize resource utilization by assigning queries to existing VMs using maximized VM capacity, as shown in (1), where n denotes the number of created VMs; m indicates the number of queries to be scheduled; L_i is the required resource of query; and x_{ij} denotes whether $Query_i$ is assigned to VM_j (if $Query_i$ is assigned to VM_j, $x_{ij} = 1$; otherwise, $x_{ij} = 0$). **Objective B** aims to save resource cost by executing queries on VMs with lower cost first to reduce load on VMs with higher cost to allow VM termination, as shown in (2), where C_j is the cost of VM_j (modeled on an hourly basis) and y_i denotes whether VM_j is to be terminated (if VM_j is to be terminated, $y_j = 0$; otherwise, $y_j = 1$). **Objective C** aims to reduce VM runtime for cost saving purpose and to reduce query response time for better performance by executing queries at the earliest time, as shown in (3), where s_i is the starting time of $Query_i$.

$$A = max(\sum_{j=1}^{n}\sum_{i=1}^{m}(L_i * x_{ij})) \quad (1)$$

$$B = max(-\sum_{j=1}^{n}(C_j * y_j)) \quad (2)$$

$$C = max(-\sum_{i=1}^{m} s_i) \quad (3)$$

The importance of above objectives contributing to the optimization problem is A > B > C. Combining the three individual objectives, we formulate the Objective D as shown in (4), subjects to constraints (5)-(16). The combination leads to a standard lexicographic optimization problem [9]. Coefficient F_0 and F_1 are given accordingly to Objective A and Objective B to guarantee the aggregated problem with maximization of all individual objective functions is consistent to the original problem, as shown in (17) and (18). The symbols of the ILP model are defined in Table I.

Objective:

$$D = \max(F_0 * \sum_{j=1}^{n}\sum_{i=1}^{m}(L_i * x_{ij}) - F_1 * \sum_{j=1}^{n}(C_j * y_j) - \sum_{i=1}^{m} s_i) \quad (4)$$

Subject to:

$$\sum_{i=1}^{m}(L_i * x_{ij}) \le CP_j, \forall i \in m, j \in n \quad (5)$$

$$x_{ij} = \{0|1\}, \forall\, i \in m;\ j \in n \quad (6)$$

$$b_{ik} + b_{ki} \le 1, \forall\, i, k \in m \quad (7)$$

$$b_{ik} = \{0|1\}, \forall\, i, k \in m \quad (8)$$

$$b_{ik} + b_{ki} - x_{ij} - x_{kj} \ge -1, \forall\, i, k \in m;\ j \in n \quad (9)$$

$$s_i - s_k + F_2 * b_{ik} \le F_2 - e_{ij}, \forall\, i, k \in m;\ j \in n \quad (10)$$

$$s_i + F_3 * x_{ij} \le F_3 + D_i - e_{ij}, \forall i \in m, j \in n \quad (11)$$

$$C_{ij} * x_{ij} \le B_i, \forall i \in m, j \in n \quad (12)$$

$$\sum_{j=1}^{n} x_{ij} \le 1, \forall i \in m, j \in n \quad (13)$$

$$y_j \ge x_{ij},\ \forall\, i \in m, j \in n \quad (14)$$

$$y_j \ge y_{j+1}, \forall\, j \in n \quad (15)$$

$$y_j = \{0|1\}, \forall\ j \in n \quad (16)$$

VM capacity constraints: as shown in (5), where x_{ij} is a binary variable indicating whether $Query_i$ is scheduled to VM_j, as shown in (6), CP_j denotes the available capacity of VM_j before the maximum deadline of the m queries. Constraint (5) ensures that the overall resources required by queries do not exceed the available capacity of VM_j.

Query deadline constraints: as shown in (7) to (11), where b_{ik} is a binary variable as shown in (8), which denotes whether $Query_i$ is executed before $Query_k$, if it is, $b_{ik} = 1$;

otherwise, $b_{ik} = 0$. Constraint (7) is used to ensure the unique execution order of $Query_i$ and $Query_k$ by allowing maximum one of b_{ik} and b_{ki} to be 1. Constraint (9) restricts that $Query_i$ must be executed either before $Query_k$ or after $Query_k$when they are executed on the same VM_j, which is derived from non-linear constraint (19). Constraint (19) ensures that if $Query_i$ and $Query_k$ are scheduled on VM_j, either $b_{ik} = 1, b_{ki} = 0$ ($Query_i$ executes before $Query_k$) or $b_{ik} = 0, b_{ki} = 1$ ($Query_i$ executes after $Query_k$). Constraint (10) is derived from non-linear constraint (20), which is used to restrict if $b_{ik} = 1$, $Query_i$ should finish before $Query_k$ starts execution, where e_{ij} denotes the execution time of $Query_i$ on VM_j and F_2 is a sufficient large constant satisfying (21). Constraint (11) is used to ensure that if $Query_i$ starts execution on VM_j at s_i, it should finish before its deadline D_i, derived from non-linear constraint (22), where F_3 is a sufficient large constant satisfying (23).

TABLE I. DEFINITION OF SYMBOLS

Symbol	Definition
i	A specific query, $Query_i$
k	A specific query, $Query_k$
j	A specific VM, VM_j
m	The set of queries
n	The set of VMs
L_i	The required resources of a query
x_{ij} / x_{kj}	It indicates whether $Query_i$/$Query_k$ is assigned to VM_j
C_j	The cost of VM_j
y_j	It indicates whether a VM_j is to be terminated
s_i / s_k	The starting execution time of $Query_i$/$Query_k$
CP_j	The available capacity of VM_j
b_{ik} / b_{ki}	It indicates whether $Query_i$ executes before $Query_k$
e_{ij}	The execution time of $Query_i$ on VM_j
D_i	The deadline of $Query_i$
B_i	The budget of $Query_i$
C_{ij}	The cost of executing $Query_i$ on VM_j
z_j	It indicates whether a VM_j is to be created

Query budget constraint: as shown in (12), where C_{ij} is the cost of executing $Query_i$ on VM_j. Constraint (12) is used to guarantee that execution cost of $Query_i$ on VM_j do not exceed its budget B_i.

Query scheduling times constraint: as shown in (13), which indicates that query can be either scheduled in Phase 1 ($\sum_{j=1}^{n} x_{ij} = 1$) or not ($\sum_{j=1}^{n} x_{ij} = 0$). If there is any query not scheduled in Phase 1, it is to be scheduled in Phase 2.

VM termination constraints are shown in (14) to (16), where y_j is a binary variable denotes whether VM_j is to be terminated, as shown in (16). Constraint (14) is used to guarantee when VM_j is to be terminated, a query should not be assigned to it, which constrains when $y_j = 0$, x_{ij} must be 0 ($Query_i$ should not be assigned to VM_j); when $y_j = 1$, x_{ij} can be 0 or 1 ($Query_i$ can either be assigned to VM_j or not). Constraint (15) is used to provide the priority of utilizing created VMs. Created VMs are input to ILP in a cost ascending list. Thus, (15) can allow the scheduler to use VMs with cheaper cost first, which allows load decreasing on VMs with higher cost to terminate VMs when they become idle for cost saving. If VMs have the same price, the query scheduler first uses VMs in front of the VM list. Moreover, the scheduler checks periodically whether any VM is idle. If there is an idle VM, the scheduler further checks whether the VM is reaching the end of its billing period. If it is, the scheduler releases the VM to serve the objective of cost minimization.

$$F_0 = max(B + C) - min(B + C) + 1 \quad (17)$$

$$F_1 = max(C) - min(C) + 1 \quad (18)$$

$$\begin{matrix} x_{ij} = 1 \\ x_{kj} = 1 \end{matrix} \overset{either}{\Longrightarrow} \begin{cases} b_{ik} = 1,\ b_{ki} = 0 \\ b_{ik} = 0, b_{ki} = 1 \end{cases}, \forall\, i, k \in m, j \in n \quad (19)$$

$$b_{ik} = 1 \Rightarrow s_i + e_{ij} \leq s_k, \forall\, i, k \in m, j \in n \quad (20)$$

$$F_2 \geq max(s_i + e_{ij} - s_k) + 1, \forall\, i, k \in m, j \in n \quad (21)$$

$$s_i * x_{ij} \leq D_i - e_{ij}, \forall\, i \in m, j \in n \quad (22)$$

$$F_3 \geq max(s_i + e_{ij} - D_i) + 1, \forall\, i \in m, j \in n \quad (23)$$

Phase 2 of scheduling starts to scale resources up by creating VMs to execute queries that are not scheduled in Phase 1. The objective is to minimize VM cost with guaranteed SLAs in query execution, as shown in (24). Objective E is subjected to all constraints the same as in Phase 1, except constraint (13). Changed constraint is shown in (25), which is used to restrict that $Query_i$ has to be scheduled to one of the newly created VMs for execution to avoid SLA violation. We use a greedy algorithm to decide the initial number of VMs of each VM type to input to Phase 2 of ILP algorithm. This ensures the capacity of input VMs is close to the required capacity of the optimal solution to avoid ILP taking long time to find a solution, which greatly reduces the algorithm running time of ILP. Given that input VMs obtained from the greedy algorithm have sufficient capacity to execute queries, to serve the cost minimization objective, ILP decides whether to create each of input VMs to execute queries, indicated by z_j. If $z_j = 1$, ILP creates input VM_j; otherwise, ILP does not create the VM. After Phase 1 and Phase 2, all queries are successfully scheduled to VMs for execution.

$$E = min \sum_{j=1}^{n} (C_j * z_j) \quad (24)$$

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i \in m, j \in n \quad (25)$$

Based on the above formulation of the scheduling problem, we build the ILP scheduling algorithm. ILP algorithm first gets all accepted queries for each requested BDAA. In Phase 1, ILP schedules queries to existing VMs running the BDAA to maximize VM utilization and execute queries with SLA guarantees. Meanwhile, ILP tries to scale down VM resources to save resource cost and executes queries at the earliest time to achieve better performance. Afterwards, ILP takes scheduling actions based on returned scheduling solution. After Phase 1, if there are queries not scheduled, Phase 2 generates scheduling solutions by scaling up VM resources to execute queries with minimized resource cost and SLA guarantees. Then, ILP takes actions based on the returned scheduling solutions of Phase 2.

2) Adapative Greedy Search (AGS) scheduling algorithm

We propose the AGS scheduling algorithm, inspired by previous works [5, 7], which aims to find a cost-effective VM configuration to execute queries while satisfying SLAs of queries by applying a greedy search heuristic. For each BDAA, in Phase 1 of scheduling, the AGS algorithm first gets queries requesting the BDAA and the created VMs running

the requested BDAA. If the BDAA is requested for the first time, one initial VM is created. AGS first tries to use all existing VMs to execute accepted queries, as shown in Lines 1-9 of the pseudo code below. AGS schedules all queries based on the urgency of deadline, which is represented by Scheduling Delay (SD). SD is the difference between deadline and expected finish time of the query. AGS first sorts queries based on SD in an ascending order for scheduling; then, AGS tries to assign each query to a VM that can satisfy its SLAs and gives it the Earliest Starting Time (EST). We name this scheduling method as the SD-based method.

AGS scheduling algorithm

```
Input: accepted queries and current VM configuration
Output: query scheduling solution and new VM configuration
1: for each BDAA ∈ requested BDAA list
2:  queries ← get accepted query list of BDAA
3:  VMs ← get running VM list of BDAA
4:    if size(queries) > 0
5:      create initial VM for BDAA if it is firstly requested
6:      queries ← sort queries based on SD
7:      for each query ∈ queries
8:        do Phase 1 schedule to assign query to VM_EST that can
satisfy its SLAs and update EST of VM_EST
9:      end for
10:     queries ← get queries not scheduled if they exist
11:     if queries contains at least one query do Phase 2 schedule
12:       CMs ← get all possible configuration modifications
13:       continue_SEARCH ← true
14:       iteration_N ← 0
15:       iteration_2N ← 0
16:       Cost(AGS)_cheapest ← a constant which is sufficient large
17:       while continueSearch is true or iteration_2N > 0
18:         iteration_N++
19:         iteration_2N--
20:         for iterator ← CMs; iterator has next CM
21:           CM_1 ← get next CM from iterator
22:           while iterator has next CM
23:             CM_2 ← get next CM from iterator
24:             C_1 ← get new VM configuration by applying CM_1
25:             C_2 ← get new VM configuration by applying CM_2
26:             Cost(AGS)_1 ← calculate cost of C_1 based on SD
27:             Cost(AGS)_2 ← calculate cost of C_2 based on SD
28:             C ← choose from C_1 and C_2 for cheaper Cost(AGS)
29:             Cost(AGS)_C ← cheaper Cost(AGS)
30:           end while
31:         end for
32:         if Cost(AGS)_C < Cost(AGS)_CHEAPEST,
33:           C_CHEAPEST ← C;
34:           Cost(AGS)_CHEAPEST ← Cost(AGS)_C
35:         end if
36:       else
37:           continue_SEARCH ← false
38:           iteration_2N ← 2 * iteration_N
39:       end else
40:     end while
41:       adopt C_CHEAPEST and take according scheduling actions
42:   end if
43:  end if
44: end for
```

After Phase 1, if there still are some queries not scheduled due to QoS constraints on deadline and budget, AGS starts Phase 2 of scheduling to find a cost-effective VM configuration to create new VMs for query execution, as shown in Lines 10-40. The set of all possible VM configurations for a requested BDAA is represented by a directed acyclic graph, $Configurations = (N, E)$, where N is a set of nodes and E is a set of edges. A node is a possible VM configuration C_i. An edge (C_i, C_{i+1}) indicates VM configuration C_{i+1} can be obtained from VM configuration C_i by applying a Configuration Modification (CM). The number of VM types available in the AaaS platform determines the number of possible CMs. The CMs are adding the cheapest VM, adding a more expensive VM, adding a next more expensive VM if it exists, till adding the most expensive VM.

Firstly, AGS gets all possible CMs and iteratively apply each CM on current resource configuration to generate a new configuration. Then, the cost manager calculates $Cost(AGS)_1$ and $Cost(AGS)_2$ of resource configurations C_1 and C_2 in the following way: for each configuration, the scheduler applies the SD-based method to schedule queries. Then the cost manager calculates the cost of the scheduling solution, which is the sum of resource cost and penalty cost. The penalty cost is set to a sufficiently high value that generates high $Cost(AGS)$ for SLA violations. This causes AGS selecting a cheaper resource configuration without violating SLAs of queries. In each search iteration, AGS selects the configuration C that gives cheapest $Cost(AGS)_c$. Then, AGS compares $Cost(AGS)_c$ with the cheapest $Cost(AGS)_{cheapest}$ generated thus far. If the $Cost(AGS)_c$ is cheaper, the cheapest configuration $C_{cheapest}$ is set as C while the cheapest cost $Cost(AGS)_{cheapest}$ is set as $Cost(AGS)_c$. AGS scheduling algorithm continues these steps until it finds the first local optimal configuration using N iterations ($iteration_N$). Afterwards, it continues exploring for another 2N iterations ($iteration_{2N}$) to see whether it can find another local optimal configuration that can give a cheaper $Cost(AGS)$. The cheapest configuration found after these 3N iterations is chosen. Finally, the scheduling solutions are adopted to execute queries, as shown in Line 41.

3) Adaptive ILP (AILP) scheduling algorithm

ILP can find the optimal solution for the scheduling problem. However, as an optimization algorithm, the Algorithm Running Time (ART) of ILP increases quickly as the input size grows, which might lead to deadline violations when ILP takes a long time to find scheduling solution. In case that an unexpected large number of queries arrive, to avoid the risk of violating deadline in such situation, we integrate AGS and ILP to form a new scheduling algorithm, which is AILP.

AILP works in the following way: it first utilizes ILP to make scheduling decision and specifies a timeout for ILP to give scheduling solution to limit the ART of ILP. When timeout is reached, if feasible integer linear programming solution is found (which may not be the optimal one), ILP returns the suboptimal solution. If no feasible solution is found, ILP only returns the timeout. After the scheduling of ILP, if there is any query that is not successfully scheduled,

AILP utilizes AGS as the alternative scheduling algorithm to make scheduling decision to avoid SLA violations. The reason that AILP utilizes ILP at the first place to give scheduling decisions is to maximize profit and guarantee SLAs. Only if timeout of ILP is reached, to avoid deadline violations that can lead to user dissatisfaction and additional penalty cost, AILP utilizes AGS as an alternative scheduling algorithm. In this way, AILP serves as the best solution to make scheduling decisions for the purposes of profit maximization and SLA guarantee while overcoming the ART limitation of ILP, which is utilized as the scheduling algorithm for the AaaS platform.

IV. Performance Evaluation

To evaluate the proposed admission control and scheduling algorithm, we build the framework of AaaS in CloudSim [10], which is a discrete event simulator. CloudSim enables repeatable and controllable experiments and offers comprehensive environment for resource scheduling and provisioning studies. We conduct experiments on the AaaS framework to evaluate the effectiveness and efficiency of the proposed admission control and resource scheduling algorithm for SLA guarantee, profit enhancement, and cost saving. ILP and AGS are utilized as baseline algorithms for the evaluation of AILP scheduling algorithm.

A. Resource Configuration

We simulate a datacenter that consists of 500 physical nodes. Each node has 50 CPU cores, 100GB memory, 10TB storage, and 10GB/s network bandwidth. We simulate five types of memory optimized VMs, which are based on Amazon EC2 VM model and managed by the resource manager. The VMs are r3.large, r3.xlarge, r3.2xlarge, r3.4xlarge, and r3.8xlarge, as shown in Table II [11]. VMs in Amazon EC2 are charged by an hourly basis with unit of dollar/hour. The unit for memory is GiB while for storage is GB.

TABLE II. VM Configuration

Type	vCPU	ECU	Memory	Storage	Cost
r3.large	2	6.5	15	32 SSD	0.175
r3.xlarge	4	13	30.5	80 SSD	0.350
r3.2xlarge	8	26	61	160 SSD	0.700
r3.4xlarge	16	52	122	320 SSD	1.400
r3.8xlarge	32	104	244	640 SSD	2.800

B. Workload

The workload generated in this study is based on the Big Data Benchmark [12], which runs query applications using different data analytic frameworks on large datasets with given data size, data type, and data location. The benchmark uses Amazon EC2 VMs and details query processing time and the VM configurations. Based on the information, resource requirements of queries requesting different BDAAs under different resource configurations are modeled in CloudSim. Each query request contains the following information: **Query submission time** is generated using a Poisson process with 1 minute mean Poisson arrival interval. **Query class** contains 4 types, which are: scan query, aggregation query, join query, and User Defined Function (UDF) query. **BDAAs**: in our experiment, 4 types of BDAAs are considered, which are built on Impala (disk) ($BDAA_1$), Shark (disk) ($BDAA_2$), Hive ($BDAA_3$), and Tez ($BDAA_4$). **Query resource** is modeled based on the resource requirements of queries. As the performance of queries vary, we model 10% performance variation by introducing a variation coefficient [13], which is generated from a Uniform Distribution with upper bound 1.1 and lower bound 0.9. **Query user**: we simulate 50 users who submit queries and request for BDAAs. **Query deadline** is generated as the benchmark does not contain QoS requirements on deadline (the same for budget). Two types of deadline are considered, tight deadline and loose deadline. Tight deadline is generated using a Normal Distribution (3, 1.4), where 3 represents that the mean deadline of a query is 3 times of its processing time and 1.4 represents the standard deviation. Loose deadline is generated similarly using a Normal Distribution (8, 3) [14]. **Query budget** contains tight budget and loose budget. Tight budget is generated using a Normal Distribution (3, 1.4) while loose budget is generated using a Normal Distribution (8, 3).

C. Result Analysis

We generate an approximate 7 hours query workload that contains 400 queries. The requested BDAAs by queries are managed by the BDAA manager. We use lp_solve 5.5 [15] as the integer linear programming solver. Queries whose QoS requirements on deadline and budget can be satisfied are admitted by admission controller. To avoid deadline violation and ensure the performance of queries, the number of queries assigned to VMs by the query scheduler is less than the number of VM cores to avoid time sharing between queries as time sharing may cause deadline violations. To evaluate the effectiveness and efficiency of the admission and scheduling algorithm, we conduct the following studies:

1) Admission control and SLA guarantee:

We conduct the real time scheduling and periodic scheduling by varying SI from 10 to 60 (unit: minute) to evaluate the effectiveness of the admission control algorithm. The results are shown in Table III, where SQN is the submitted query number, AQN is the accepted query number, and SEN is the successfully executed query number.

TABLE III. Query Number Information

Number	Real Time	Periodic					
		10	*20*	*30*	*40*	*50*	*60*
SQN	400	400	400	400	400	400	400
AQN	336	317	299	287	274	261	252
SEN	336	317	299	287	274	261	252

We calculate the acceptance rate based on the obtained results. For real time scheduling, the rate is 84.0%, while for periodic scheduling of SI from 10 to 60, the rate is 79.3 %, 74.8 %, 71.8 %, 68.5 %, 65.3 %, and 63. 0%. As the SI increases, the acceptance rate of queries decreases due to deadline constraints. Short SI is recommended as more queries can be admitted to increase user satisfaction and enlarge market share. We can see that all accepted queries are successfully executed with SLA guarantees, which helps to

increase the reputation of AaaS providers. Obtained results show the effectiveness of the admission control algorithm in query admission.

2) Cost saving and profit enhancement

In this study, we evaluate the efficiency of AILP, ILP, and AGS scheduling algorithms in cost saving and profit enhancement. AILP is an integration of AGS and ILP, it first utilizes ILP to find scheduling solution and only switches to AGS when no feasible solution is returned by ILP within specified scheduling timeout to avoid potential deadline violations. When timeout of AILP is reached, we record the data to show the contribution of ILP and AGS in generating the scheduling solution of AILP. For periodic scheduling using ILP, we only record the scheduling solutions that are given within the SI. As scheduling solutions exceeding the SIs are not applicable, from this paragraph on, we do not include ILP in algorithm comparison for simplicity purpose. Obtained results are shown in Fig. 2. We note that for real time scheduling and periodic scheduling with SI=10 and SI=20, ILP algorithm can schedule queries within the specified timeout. This indicates that scheduling decisions of AILP for these SIs are optimal and are given by ILP. For SI=30 and SI=40, in some SIs, scheduling timeout of ILP is reached, which indicates the solution given by AILP might be sub-optimal. For SI=50 and SI=60, AGS is utilized along with ILP to contribute to the scheduling solution of AILP. Based on the obtained results, we can know that, for real time scheduling, the resource cost of AILP is 7.3% less than AGS, while for periodical scheduling with SI from 10 to 60, resource cost generated by AILP is 11.3%, 9.3%, 4.8%, 4.4%, 5.4%, and 4.3% less than AGS.

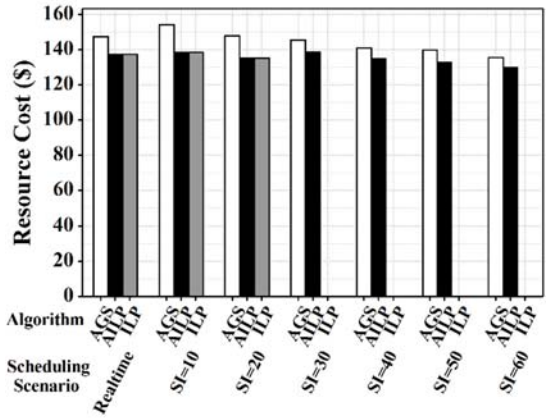

Figure 2. Resource Cost of AGS, AILP, and ILP

The resource configuration to execute the query workload is shown in Table IV. Results show that, for all scheduling scenarios, AILP can efficiently reduce the numbers of VMs that are used. The observation is consistent with the Objective B and Objective E, as shown in (2) and (24), and constraint (15). ILP uses cheaper VMs at first if they are available; then, for the same VM type, ILP first uses VMs in the front of the VM queue. This leads to cheaper resource cost and higher resource utilization, which also leads to less VM creations. Creating less VMs is beneficial as it takes several minutes to boot VMs and make VMs online. We use 97 seconds as VM configuration time in our experiment [16]. Creating more VMs will cause more delay to serve requests, which may cause deadline violations and lead to higher resource cost. Thus, less VM creations can better benefit our objective of SLA guaranteeing and cost saving. Results show that the VM types utilized by AILP and AGS are r3.large and r3.xlarge while other types of r3.2xlarge, r3.4xlarge, and r3.8xlarge are not utilized. We find the reason that, indicated by Table II, there is no pricing advantages to use VMs with larger capacity as the capacity of VM increases, the price increases proportionally. Since VMs is not always fully utilized and creating VMs with higher capacity leads to higher resource cost, expensive VMs are not given the priority for utilization.

TABLE IV. RESOURCE CONFIGURATION

Scheduling Scenario		AGS	AILP
Real Time		58 ∗ r3.large	23 ∗ r3.large
Periodic	***SI=10***	48 ∗ r3.large	23 ∗ r3.large
	SI=20	27 ∗ r3.large	22 ∗ r3.large
	SI=30	32 ∗ r3.large	22 ∗ r3.large
	SI=40	28 ∗ r3.large, 2 ∗ r3.xlarge	22 ∗ r3.large
	SI=50	28 ∗ r3.large	17 ∗ r3.large, 2 ∗ r3.xlarge
	SI=60	21 ∗ r3.large, 4 ∗ r3.xlarge	16 ∗ r3.large, 2 ∗ r3.xlarge

Profit gained by AILP and AGS is shown in Fig. 3. We notice that for both real time scheduling and periodic scheduling, AILP performs significantly better than AGS in profit enhancement. For real time scheduling, profit gained by AILP is 11.4% more than AGS. For periodic scheduling of SI from 10 to 60, profit gained by AILP is 19.8%, 15.2%, 7.9%, 6.7%, 8.2%, and 6.1% more than AGS.

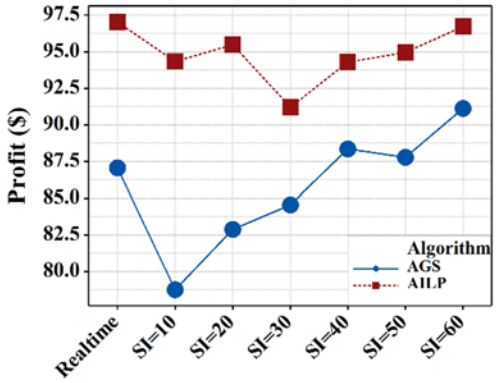

Figure 3. Profit of AILP and AGS

For real time scheduling, profit created by AILP and AGS is relatively high as more queries are admitted that leads to higher query income. The profit decreases when the real time scheduling is switched to the periodic scheduling with SI=10 as more queries are rejected, which decreases the query income. With the continuous increase of SI, as more queries are scheduled during each SI, AGS and AILP can make better scheduling decisions to provision resources and schedule queries to VMs, which increases the profit. For AGS with SI from 10 to 60, we notice the overall increasing trend of profit is relatively obvious comparing to AILP. For AILP, we notice

that profit increases when SI increases from 10 to 20. However, when SI increases from 20 to 30, the profit drops as AILP returns sub-optimal solutions in some SIs when timeout is reached. We further notice that when SI increases from 30 to 40, the profit starts increasing again, which indicates that AILP can give higher cost saving to create higher profit even though the query income is decreased when more queries are rejected. When SI of AILP increases from 40 to 50 and from 50 to 60, the profit keeps increasing as more queries are scheduled that leads to better scheduling decisions. For SI=50 and SI=60, the solutions of AILP are given by integration of ILP and AGS, as for some SIs, AGS are utilized when feasible solutions are not given by ILP, which weakens the profit enhancement and cost saving advantages of ILP.

As shown in Fig. 4, for all scheduling scenarios, the median resource cost for AILP is $135.3 while for AGS is $145.4, which shows AILP gives 7.5% higher cost saving regarding median resource cost. The median profit made by AILP is $95.0 while by AGS is $87.0, which shows AILP creates 9.2% higher median profit than AGS. Results also show that the mean resource cost generated by AILP is $135.3, which is 6.7% less than AGS while the mean profit created by AILP is $94.9, which is 10.6% higher than AGS.

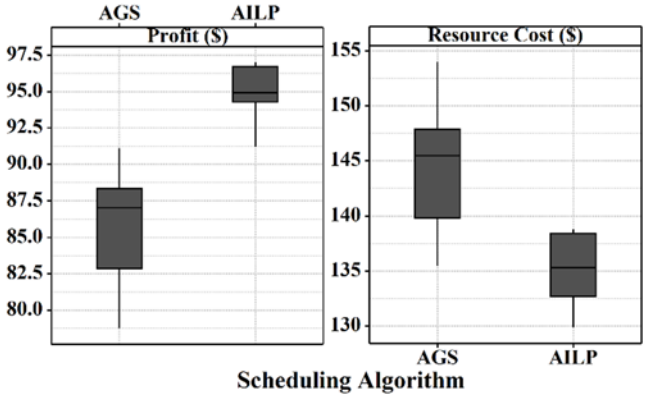

Figure 4. Profit and Resource Cost of AILP and AGS

All of the obtained results show that AILP generates less resource cost and creates higher profit comparing to AGS, which proves the efficiency of AILP in achieving the cost saving and profit enhancement objectives. For real time scheduling, the benefit is that more queries can be accepted for higher profit, while the drawback is that frequent scheduling consumes more computing resources. Thus, real time scheduling is not applicable when load of queries is high. For periodic scheduling, the more queries are collected, the better scheduling decisions can be made to save more resource cost and create higher profit. However, the drawback is that the longer the SI is, the more queries are rejected due to deadline constraints. This causes customer dissatisfaction due to higher request rejection rate, which also leads to reduction of market share. Therefore, long SI is not recommended even more resource cost can be saved. In order to achieve higher cost saving and profit enhancement, while at the same time accept more queries to serve more users for larger market share and higher user satisfaction, periodic scheduling with SI=20 is the best solution to execute the query workload.

We further study the performance of AILP and AGS in profit enhancement and cost saving for different BDAAs using SI=20. Results are shown in Fig. 5. We notice that resource cost and profit vary for different BDAAs, which is caused by variation of the number of accepted queries and their required resources. For different queries, query execution time can vary from minutes to hours. Resource cost generated by AILP is 1.9%, 2.4%, 15.5%, and 3.3% less than AGS for $BDAA_1$, $BDAA_2$, $BDAA_3$, and $BDAA_4$. Accordingly, the profit created by AILP is 3.5%, 4.3%, 26.2%, and 4.8% higher than AGS. All the obtained results show AILP can generate less resource cost and create higher profit for each BDAA that further prove that AILP can better serve the cost saving and profit enhancement objectives.

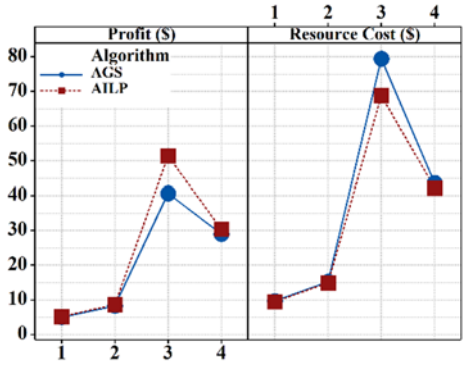

Figure 5. Profit and Resource Cost of BDAAs for SI=20

3) Study of algorithm performance based on C/P metric

We use the C/P metric [5] to evaluate the performance of AILP and AGS. C/P is the quotient of resource cost and workload running time. Smaller C/P value indicates better performance of the scheduling algorithm. We can see that AILP has smaller C/P values for all scheduling scenarios, which indicates that AILP performs better than AGS, as shown in Fig. 6.

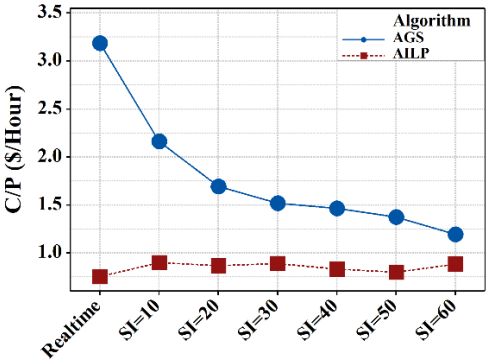

Figure 6. C/P of AILP and AGS

We use periodic scheduling with SI=20 as an example. For SI=20, results show that to execute the query workload, AILP uses 156.5 hours that is 78.9% longer than AGS and generates $135.3 resource cost that is 9.3% less than AGS. This leads to a smaller C/P value of 0.9 for AILP, comparing to larger C/P value of 1.7 for AGS. For AGS, we notice that the trend of C/P keeps decreasing while SI increases, which indicates that

AGS performs better when more queries are scheduled in each SI. For AILP, the trend of C/P fluctuates as the scheduling solution returned by AILP is an integration of ILP and AGS.

4) Study of ART

We record the ART for AILP and AGS to execute the query workload, as shown in Fig. 7. For real time scheduling, a query is scheduled based on its arrival. There is one query that is scheduled each time by AGS and AILP. Thus, ART is not a limitation factor. For periodic scheduling, variation of SI leads to variation of the number of queries to be scheduled in each SI. To avoid deadline violations when the number of queries is large, a scheduling timeout is set to limit the ART. The maximum value of the timeout is limited to 90% of SI to ensure that sufficient time is left for AGS to give scheduling solution when no feasible solution is given by ILP.

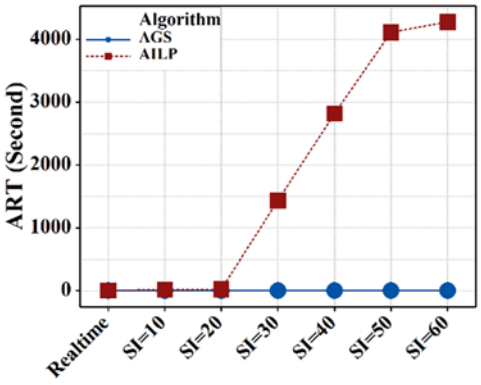

Figure 7. ART of AILP and AGS

We made efforts to reduce the ART of ILP by utilizing two greedy algorithms to decide the input of VMs to Phase 1 and Phase 2 of ILP scheduling algorithm, which ensures that the given VMs have sufficient capacity to execute queries; meanwhile, the given number of VMs is close to the number of VMs of the optimal solution. In this way, ILP does not need to search large solution space, which greatly reduces the ART of ILP. For different query workload, SI can be adjusted to a suitable value based on the arrival rate of queries to allow AILP algorithm generating scheduling solutions to better achieve cost saving and profit enhancement objectives.

Results show that ART_{AILP} is longer than ART_{AGS} for all scheduling scenarios. For periodical scheduling, AGS takes milliseconds to give a scheduling decision for each SI and ART_{AGS} increases when SI increases. However, the increasing rate of ART_{AGS} is not as rapid as ART_{AILP}. For AILP, the increasing rate of ART_{AILP} decreases while SI increases as the ART cannot exceed the scheduling timeout, which is a constant value. Obtained results show that ART is not the limiting factor for AILP to serve as the scheduling algorithm of the AaaS platform as it can give scheduling solution within each SI. However, it is not recommended to use long SI. As the longer the SI is, the more requests with tight deadline are rejected by the admission controller, which is not consistent with the profit maximization objective of the AaaS platform. Moreover, it also decreases the user satisfaction and reduces the market share of AaaS providers.

V. Related Work

Our research focuses on provisioning effective and efficient algorithms to support the AaaS platform to deliver on-demand AaaS for various domains of users with SLA guarantees and at lower cost in order to enhance profit, enlarge market share, and improve user satisfaction. We compare our work with the most related works with respect to various parameters, as summarized in Table V.

TABLE V. Related Work

Parameter	Related Work						Our Work
	[4]	*[5]*	*[7]*	*[17]*	*[18]*	*[19]*	
AaaS	Y	N	N	Y	N	N	Y
Cost saving	Y	Y	Y	Y	Y	Y	Y
Profit enlarging	N	N	N	N	N	N	Y
SLA guarantee	Y	N	Y	Y	Y	N	Y
Admission control	N	N	Y	N	N	N	Y
Budget	N	N	N	N	Y	Y	Y
Deadline	N	N	N	Y	Y	Y	Y
Scalable and elastic resource provision	N	Y	Y	Y	N	Y	Y

Y: covered; N: not covered

Sun et al. [4] propose a general-purpose analytic framework to provision cost-effective analytic solution with multi-tenancy support. They propose an SLA customization mechanism to satisfy diverse QoS requirements of tenants. However, their work does not consider admission control; thus, SLAs are at risk of violations. Moreover, they do not address resource scheduling algorithms with SLA guarantees on deadline and budget and also do not address profit enhancement purpose of AaaS providers.

Mian et al. [5] focus on resource provisioning for data analytic workloads in a public Cloud that aims to determine the most cost-effective resource configuration using greedy search heuristic. Their work schedules queries for one application, which does not serve general data analytic purpose. Moreover, the SLA they considered is average query response time and SLA violation is allowed for a cheaper resource cost, which is different from our SLA guaranteeing perspective. We believe that SLA violations can significantly decrease user satisfaction and reputation of AaaS providers. Therefore, SLA violations should be avoided.

Garg et al. [7] manage resources for heterogeneous workload in a datacenter. They focus on resource scheduling problem for mixed workloads instead of data analytic workloads. They propose an admission control and resource scheduling algorithm, which considers deadline-constrained scheduling while we consider budget-constrained scheduling as well, which is the market feature of delivering analytics as a service in Cloud computing environments. Zulkernine et al. [17] propose the conceptual architecture of Cloud-based Analytics as a Service, which does not consider admission control. Their work focuses on data analytics for workflow applications.

Scheduling and resource provisioning for MapReduce tasks is a well-explored area, which targets at a specific application model and does not serve general data analytic

purpose. Alrokayan et al. [18] propose cost-aware and SLA-based algorithms to provision Cloud resources and schedule MapReduce tasks under budget and deadline constraints. They adopt Lambda architecture and focus their current work on the batch layer. Mattess at al. [19] propose an algorithm to dynamically provision resources to meet soft deadline of MapReduce tasks while minimizing the budget. Both of the above works do not consider admission control for SLA guaranteeing purpose.

There are some works focus on delivering specific analytics as a service and provisioning query processing techniques, which give support to the AaaS platform and enrich the BDAAs that the AaaS platform can provide to users. Chen et al. [20] address the challenges in delivering continuous analytics as a service to analyze real-time events. Barga et al. [21] propose Daytona to offer scalable computation for large scale data analytics, which focuses on spreadsheet applications. Agarwal at al. [22] propose BlinkDB, which enables approximate query processing on data samples of large datasets with bounded response time and bounded errors.

VI. Conclusions and Future Work

Data analytics has significant benefits in decision making and problem solving for various domains. To support the AaaS platform that offers data analytics as a service to various domains of users as consumable services, our research focuses on proposing effective and efficient admission control and resource scheduling algorithms. The aim is to allow AaaS providers delivering AaaS with SLA guarantees to increase market share, improve reputation, and enhance profit. We have proposed the architecture and models of the AaaS platform and conducted experiments to evaluate the performance of the admission control and resource scheduling algorithm. Experiments are conducted based on different scheduling scenarios, which are non-periodic (real time) scheduling and periodic scheduling with varied scheduling intervals. Results have shown that our admission control algorithm successfully admits queries based on QoS requirements to allow scheduling algorithms giving SLA guarantees. Moreover, AILP, which is an integration of AGS and ILP, can save more resource cost and create higher profit while overcoming the limitation of ILP in ART. As part of the future work, we will (1) continue working on proposing efficient admission control and resource scheduling algorithms; (2) study the effect of application profiling in the performance of algorithms; and (3) study data sampling techniques that allow query processing on sampled datasets for quicker response time and higher cost saving.

Acknowledgments

We thank Mohsen Amini Salehi, Yun Yang, Satish Srirama, and all the members at the CLOUDS Lab for their valuable comments and suggestions to improve the work.

References

[1] 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.

[2] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in big data analytics," Journal of Parallel and Distributed Computing, vol. 74, pp. 2561-2573, 2014.

[3] http://www-03.ibm.com/software/products/en/social-media-analytics-saas

[4] X. Sun, B. Gao, L. Fan, and W. An, "A cost-effective approach to delivering analytics as a service," in Proc. of 2012 IEEE 19th International Conference on Web Services (ICWS), pp. 512-519, 2012.

[5] R. Mian, P. Martin, and J. L. Vazquez-Poletti, "Provisioning data analytic workloads in a Cloud," Future Generation Computer Systems, vol. 29, pp. 1452-1458, 2013.

[6] M. Benichou, J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent, "Experiments in mixed-integer linear programming," Mathematical Programming, vol. 1, pp. 76-94, 1971.

[7] S. K. Garg, S. K. Gopalaiyengar, and R. Buyya, "SLA-based resource provisioning for heterogeneous workloads in a virtualized Cloud datacenter," in Proc. of the 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), pp. 371-384, 2011.

[8] R. Benayoun, J. De Montgolfier, J. Tergny, and O. Laritchev, "Linear programming with multiple objective functions: Step method (STEM)," Mathematical Programming, vol. 1, pp. 366-375, 1971.

[9] H. Isermann, "Linear lexicographic optimization," Operations-Research-Spektrum, vol. 4, pp. 223-228, 1982.

[10] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of Cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, pp. 23-50, 2011.

[11] http://aws.amazon.com/ec2/instance-types/

[12] https://amplab.cs.berkeley.edu/benchmark/

[13] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the Cloud: observing, analyzing, and reducing variance," in Proc. of the VLDB Endowment, vol. 3, pp. 460-471, 2010.

[14] R. N. Calheiros and R. Buyya, "Cost-effective provisioning and scheduling of deadline-constrained applications in hybrid Clouds," in Proc. of the 13th International Conference on Web Information System Engineering (WISE), pp. 171-184, 2012.

[15] http://lpsolve.sourceforge.net/5.5/

[16] M. Mao and M. Humphrey, "A performance study on the VM startup time in the Cloud," in Proc. of 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), pp. 423-430, 2012.

[17] F. Zulkernine, P. Martin, Y. Zou, M. Bauer, F. Gwadry-Sridhar, and A. Aboulnaga, "Towards Cloud-Based Analytics-as-a-Service (CLAaaS) for big data analytics in the Cloud," in Proc. of 2013 IEEE International Congress on in Big Data (BigData Congress), pp. 62-69, 2013.

[18] M. Alrokayan, A. Vahid Dastjerdi, and R. Buyya, "SLA-aware provisioning and scheduling of Cloud resources for big data analytics," in Proc. of 2014 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), pp. 1-8, 2014.

[19] M. Mattess, R. N. Calheiros, and R. Buyya, "Scaling MapReduce applications across hybrid Clouds to meet soft deadlines," in Proc. of 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), pp. 629-636, 2013.

[20] Q. Chen, M. Hsu, and H. Zeller, "Experience in Continuous analytics as a Service (CaaaS)," in Proc. of the 14th International Conference on Extending Database Technology, pp. 509-514, 2011.

[21] R. S. Barga, J. Ekanayake, and W. Lu, "Project daytona: Data analytics as a Cloud service," in Proc. of 2012 IEEE 28th International Conference on Data Engineering (ICDE), pp. 1317-1320, 2012.

[22] S. Agarwal, A. P. Iyer, A. Panda, S. Madden, B. Mozafari, and I. Stoica, "Blink and it's done: interactive queries on very large data," in Proc. of the VLDB Endowment, vol. 5, pp. 1902-1905, 2012.