Profit Maximization and Time Minimization Admission Control and Resource Scheduling for Cloud-Based Big Data Analytics-as-a-Service Platforms

Yali Zhao[1(✉)], Rodrigo N. Calheiros[2], Athanasios V. Vasilakos[3], James Bailey[1], and Richard O. Sinnott[1]

[1] The University of Melbourne, Melbourne, Australia
yalizhao.alice@gmail.com,
{baileyj,rsinnott}@unimelb.edu.au
[2] Western Sydney University, Sydney, Australia
r.calheiros@westernsydney.edu.au
[3] Lulea University of Technology, Lulea, Sweden
athanasios.vasilakos@ltu.se

Abstract. Big data analytics typically requires large amounts of resources to process ever-increasing data volumes. This can be time consuming and result in considerable expenses. Analytics-as-a-Service (AaaS) platforms provide a way to tackle expensive resource costs and lengthy data processing times by leveraging automatic resource management with a pay-per-use service delivery model. This paper explores optimization of resource management algorithms for AaaS platforms to automatically and elastically provision cloud resources to execute queries with Service Level Agreement (SLA) guarantees. We present admission control and cloud resource scheduling algorithms that serve multiple objectives including profit maximization for AaaS platform providers and query time minimization for users. Moreover, to enable queries that require timely responses and/or have constrained budgets, we apply data sampling-based admission control and resource scheduling where accuracy can be traded-off for reduced costs and quicker responses when necessary. We conduct extensive experimental evaluations for the algorithm performances compared to state-of-the-art algorithms. Experiment results show that our proposed algorithms perform significantly better in increasing query admission rates, consuming less resources and hence reducing costs, and ultimately provide a more flexible resource management solution for fast, cost-effective, and reliable big data processing.

Keywords: Optimization · Service level agreement · Analytics-as-a-service · Admission control · Resource scheduling · Data sampling · Big data · Cloud computing

J. Miller et al. (Eds.): ICWS 2019, LNCS 11512, pp. 26–47, 2019.
https://doi.org/10.1007/978-3-030-23499-7_3

1 Introduction

Big data becomes an emerging trend that many organizations and companies are currently facing. A tremendous amount of data is being produced at an ever-accelerating rate from a myriad of sources such as IoT sensors, smart mobile devices, social media platforms amongst many other sources covering all aspects of society including health, commerce, government, and education. The need for advanced technologies to tackle data production, processing, and storage is clear. However, big data can only create values through the help of data analytics technologies to extract insights from such big datasets. It is fair to say that the success of many organizations, companies, and individuals lies heavily on big data analytics solutions.

Big data typically refers to massive volumes of structured, semi-structured, unstructured or real time data that exceed the processing and management capacities of traditional techniques [1]. Big data analytics is usually associated with cloud computing technologies. Cloud computing [2] provides a computing paradigm to dynamically provision resources for big data analytics solutions based on varying numbers of requests from users. As the data volumes increase to levels that exceed the storage and processing capabilities of individual computers, clouds allow to automatically and elastically provision cloud resources such as virtual machines (VMs) on-the-fly to process datasets for timely decision making as required by users and businesses. Cloud resources are typically provisioned in a pay-per-use model that enables users to only pay for the used resources. As such, big data analytics for small enterprises and individuals is possible, as they do not need to acquire expensive hardware and software directly or deal with the overheads with the management of such infrastructures.

AaaS platforms provide a way to tackle expensive resource costs and avoid long query times by leveraging automatic resource management capabilities. AaaS service delivery should ideally minimize the complexity of the resource management requirements for in depth knowledge of Big Data Analytics Applications (BDAAs) by users. It allows users from various domains to process big data with reduced times and cheaper costs based on SLA agreements between AaaS users and providers. In order to allow AaaS platforms to deliver SLA guaranteed AaaS for high user satisfactions, efficient and automatic resource scheduling is essential [3]. Resource scheduling is a core function of the AaaS platform and a central component to coordinate all the other AaaS platform components to deliver performance-oriented AaaS solutions. Our motivation in this paper is to provide optimal resource scheduling algorithms that can, on the one hand maximize the profits of AaaS providers, while on the other hand deliver AaaS to users within controllable budgets, factoring in deadline and accuracy guarantees for timely and reliable decision making.

A number of related research works [3, 11–25] have explored resource management in different perspectives and scenarios with support of various techniques in cloud environments. However, none of these works consider data sampling-based admission control and cloud resource scheduling algorithms to deliver AaaS with budget, accuracy, and deadline guarantees to support cost-efficient, reliable, and fast decision making. Resource scheduling of AaaS platforms faces several research challenges. Firstly, elastic and automatic resource provisioning is required to deal with dynamic

online queries. Such requests can be stochastic in nature and have varying demands on the underlying cloud resources. Secondly, to support queries on very large datasets that cannot be processed under limited budgets or deadlines where accuracy can be traded-off for approximate processing [4], data sampling-based scheduling can be used. Thirdly, effective admission control should be applied to only admit queries satisfying Quality of Service (QoS) requirements that meet given SLAs. Finally, profit maximization and time minimization scheduling under various constraints represent a non-trivial multi-objective optimization problem, especially for complex BDAAs. This requires accurate modelling and formulation of the optimization problem.

To tackle the above challenges, the major **contribution** of our work is providing efficient and effective admission control and resource scheduling algorithms to elastically and automatically provision cloud resources to schedule queries that serve the objectives of profit maximization for AaaS providers and query time minimizations for users while guaranteeing SLAs. The proposed algorithms apply data sampling-based admission and scheduling methods for big data processing under tight budget or deadline constraints. We formulate the resource management problem and implement the proposed admission and scheduling algorithms in the AaaS framework. To evaluate the performance of the proposed algorithms, we conduct extensive experiments. Experiment evaluations show that the proposed admission control and resource scheduling algorithms outperform the state-of-the-art algorithms in admitting more queries, creating higher profits, generating less resource costs with efficient resource configurations, and provide timely AaaS solutions with accuracy guarantees for reliable decision making under controllable budgets and tight deadlines.

2 Problem Statement

User submit queries $Querys = \{Query_1, \ldots, Query_M\}$ to a given AaaS platform. A query can be a scan query such as select patient name from patient table where patient age is greater than 30, or a more complex query such as a deep learning algorithm used to process large medical datasets to analyze the key factors that lead to diabetes. These queries request specific $BDAAs = \{BDAA_1, \ldots, BDAA_P\}$ that utilize a set of cloud resources $Resources = \{Resource_1, \ldots, Resource_N\}$ and generating resource $Costs = \{Cost_1, \ldots, Cost_N\}$. An example *BDAA* can be a medicare application. A resource can be cloud containers, storage, single or clusters of VMs.

$Query = \{QoS, CR, BDAA, CH, AC, DE\}$. *QoS* requirements of a query request contain budget: the maximum costs to run a query; deadline: the latest time to deliver query result, accuracy: the confidence interval of the query result. *CR* is the cloud resources needed to run a query. *BDAA* details a specific BDAA requested for big data analytics. *CH* details the big data characteristics, such as the data distribution, data size, data type, accuracy requirement, and data locality. *AC* indicates whether the accuracy can be traded-off for reduced times and cheaper costs by applying effective data sampling methods. *DE* represents task dependency requirements including execution logic and sequence. Big data is assumed to be pre-stored in cloud datacenters, where users pay data transfer and storage costs and hence the costs are not included in the cost models.

$BDAA = \{AT, P\}$. AT denotes the BDAA type and P is the BDAA profile. This includes mappings of the application cost, resource times required by queries including the processing and sampling time, and the resource configurations needed for queries. Obtaining application profiles of heterogeneous BDAAs typically requires expertise varied by different application domains. Therefore, reliable BDAA profiles are assumed to be maintained by third-party BDAA providers.

$Resource = \{RT, REC, NR, CC\}$. RT indicates the cloud resource type such as single or cluster of CPU optimized VMs. REC shows the cloud resource capacity including CPU, storage, memory. NR is the number of cloud resources needed to execute a query request. CC represents the cloud resource cost.

$Cost = \{RC, BC, PC, AC, PR\}$. RC is the overall cloud resource costs of the AaaS platform. BC represents the BDAA cost, which is assumed charged by BDAA providers as a constant value. PC represents the penalty cost that AaaS platforms need to pay users for SLA violation. AC represents the overall AaaS Cost (AC) that AaaS platforms charge users for utilizing AaaS, and PR represents the overall profits created by AaaS platforms. A fixed AaaS cost is assumed charged by AaaS platforms for a specific query based on its requirements of QoS, BDAA, and resource demands, and hence AC is a constant value.

The profit optimization problem to maximize the profits of the AaaS platform while minimizing the query times under various constraints is an NP-complete decision problem. The problem can be polynomially transformed to a mixed Integer Linear Programming (ILP) [5] problem with optimization formulation and modeling. We utilize the ILP modeling to formulate the multi-objective scheduling problem. AC and BC are constant values based on the cost models and PC is zero if AaaS is delivered with SLA guarantees. The profit of the AaaS platform is calculated by $PR = AC - RC - BC - PC$. Maximization of PR is transformed to minimize RC and deliver AaaS with minimized times subject to various constraints. We formulate the multi-objective Z based on a combination of individual optimization Objective X and Objective Y.

Objective $\boldsymbol{X} = minimize\ (\sum_{j=1}^{n} C_j * t_j)$, where j denotes a resource; n represents a resource set; t_j is the number of resource required for j; C_j is the unit cloud resource cost of j. Objective X targets to minimize the RC, calculated as the product of the purchased time t_j and the unit cost C_j of all cloud resources.

Objective $\boldsymbol{Y} = minimize\ (\sum_{i=1}^{m} s_i)$, where i is a query i; m represents a set of queries, and s_i represent the query start time of i. Objective Y targets at finding an optimal solution of the resource scheduling to minimize the resource costs with optimized configuration to execute queries at its earliest times. This allows all queries starting earliest times to minimize t_j to save resource costs and improve the query performances with quickest responses.

Objective $\boldsymbol{Z} = minimize\ (F_0 * \sum_{j \in n} (C_j * t_j) + \sum_{i \in m} s_i)$ is a combination of X and Y that aims to minimize RC while choosing a time-minimized scheduling plan, subject to optimization constraints (1)–(37). This combination contributes to a standard lexicographic problem [6]. The importance leading to the profit optimal scheduling solutions of individual objectives is $X > Y$. Coefficient $F_0 = max(Y) - min(Y) + 1$ is assigned to X to ensure the aggregated optimization with minimized individual

objectives is consistent to the original optimization problem where changes of Objective X dominates all the changes in Objective Y.

Query Resource Capacity Constraints guarantee that the overall cloud resource times needed to process the complete datasets on resource j should be within the available time remained on resource j, (1), where x_{ij} is a variable with binary value that represents if query i is assigned to run on j; R_{ij} is the cloud resource time needed by i to process the complete dataset using resource j; t_j is the resource time of j that is to be purchased. For a given resource, T_j represents the remained resource time of j that is purchased in the previous schedule. For resources that is newly created, T_j is negative to deduct the resource creation time of j from t_j. When i executes on j, $x_{ij} = 1$; otherwise, $x_{ij} = 0$ shown in (2).

$$\sum\nolimits_{i \in m, j \in n} \left(R_{ij} * x_{ij}\right) \leq t_j + T_j \tag{1}$$

$$x_{ij} = \begin{cases} 1, & i \text{ is assigned to execute on } j \\ 0, & \text{otherwise} \end{cases}, \forall i \in m; j \in n \tag{2}$$

Query Execution Sequence Constraints (3) ensure the unique query execution sequence as required by the optimal scheduling. b_{ik} is defined as a binary variable that defines the query execution sequence of i and k. If query i is scheduled to execute before k, $b_{ik} = 1$; otherwise, $b_{ik} = 0$.

$$b_{ik} + b_{ki} \leq 1, \forall i, k \in m \tag{3}$$

Query Dependency Constraints define query dependencies, $b_{ik} = 1$ when i is a dependent task requiring the execution results of k or a child task that can only start after k defined by (4). Query dependency constraints enable the definition of simple to complex task dependencies including bag of tasks, enterprise workflows, and scientific dataflows. For queries requiring data sampling for approximate processing, where data sampling and processing tasks are processed independently, query processing on sampled datasets can only start after data sampling finishes, as shown in (5).

$$b_{ki} = 1, \text{ if } i \text{ is a child/dependent task of } k, \forall\, i, k \in m \tag{4}$$

$$b_{ki} = 1, \text{ if } i \text{ processes data sample of } k, \forall\, i, k \in m \tag{5}$$

Query Deadline Constraints ensure scheduling solutions are generated before query deadlines for SLA guarantee purposes, as shown in Constraints (6)–(11). Constraint (6) guarantee the unique query execution sequence of i and k that allows either b_{ik} or b_{ki} to be 1. Task i must execute either before or after task k when they are assigned to the same resource j as guaranteed by (7). Non-linear relationship (12) is transformed to linear relationship (7) through linearization, which guarantees if both i and k are scheduled to execute on j, $b_{ik} = 1, b_{ki} = 0$ when i is executed before k, or $b_{ik} = 0, b_{ki} = 1$ when i is executed after k. Big M method is utilized to derive Constraint (8) from non-linear constraint (13) to guarantee i should end before k starts if

$b_{ik} = 1$. F_1 serves as a sufficient large constant that satisfies (14) [3]. Constraint (9) ensure query i finishes before its deadline D_i when executes on j at starting time s_i, derived from (15), which is non-linear constraint where F_2 serves as a sufficient large constant that satisfies (16). Non-linear relationship (17) is used to derive (10). F_3 serves as a sufficient large constant value that satisfies (18) to guarantee i finishes earlier than the purchased time of j to process the full dataset. The purchased time is accumulated from previous End of Purchased Time, EPT_j, and newly purchased time as t_j. Constraint (11) is generated based on the non-linear constraint (19). F_4 guarantees i starts at Earliest Available Time, EAT_j, of j, which satisfies (20) as a sufficient small constant. For an already created j with executing tasks, EAT_j represents the query finish time of j; otherwise, EAT_j represents the current clock time. If j has not been created, EAT_j is the sum of creation time and clock time of resource j.

$$b_{ik} = \begin{cases} 1, & i \text{ is executed before } k \\ 0, & otherwise \end{cases}, \forall i,k \in m \tag{6}$$

$$b_{ik} + b_{ki} - x_{ij} - x_{kj} \geq -1, \forall i,k \in m;\ j \in n \tag{7}$$

$$s_i - s_k + F_1 * b_{ik} \leq F_1 - R_{ij}, \forall i,k \in m;\ j \in n \tag{8}$$

$$s_i + F_2 * x_{ij} \leq F_2 + D_i - R_{ij}, \forall i \in m, j \in n \tag{9}$$

$$s_i - t_j + F_3 * x_{ij} \leq F_3 - R_{ij} + EPT_j, \forall i \in m, j \in n \tag{10}$$

$$s_i + x_{ij} * F_4 \geq F_4 + EAT_j, \forall i \in m, j \in n \tag{11}$$

$$\left.\begin{matrix} x_{ij} = 1 \\ x_{kj} = 1 \end{matrix}\right\} \overset{either}{\Rightarrow} \begin{cases} b_{ik} = 1, b_{ki} = 0 \\ b_{ik} = 0, b_{ki} = 1 \end{cases}, \forall i,k \in m, j \in n \tag{12}$$

$$b_{ik} = 1 \Rightarrow s_i + R_{ij} \leq s_k, \forall i,k \in m, j \in n \tag{13}$$

$$F_1 \geq \max(s_i + R_{ij} - s_k) + 1, \forall i,k \in m, j \in n \tag{14}$$

$$s_i * x_{ij} + R_{ij} \leq D_i, \forall i \in m, j \in n \tag{15}$$

$$F_2 \geq \max\ (s_i + R_{ij} - D_i) + 1, \forall i \in m, j \in n \tag{16}$$

$$x_{ij} = 1 \Rightarrow s_i + R_{ij} \leq t_j + EPT_j, \forall i \in m, j \in n \tag{17}$$

$$F_3 \geq \max\ (s_i - t_j + R_{ij} - EPT_j) + 1, \forall i \in m, j \in n \tag{18}$$

$$x_{ij} = 1 \Rightarrow s_i \geq EAT_j, \forall i \in m, j \in n \tag{19}$$

$$F_4 \leq min\ \left(s_i - EAT_j\right) - 1, \forall i \in m, j \in n \tag{20}$$

Data Sampling-based Query Budget Constraints guarantee that the resource cost to execute i on j is within the budget of B_i. If the entire dataset of query is to be processed

in full, as shown in (21), C_{ij} represents the execution cost of query i on j that is within B_i. Otherwise, the accuracy of query may have to be sacrificed due to time and budget constraints, as shown in (22). SC_{ij} is the cost to sample and execute i on the data samples on j. This should be less than the budget of the query as B_i. Constraint (24) ensure if the cost of j exceeds the B_i j will be eliminated from the resource pool to enable the ILP solver to reduce the solution space and improve the algorithm performance.

$$C_{ij} * x_{ij} \leq B_i, \forall i \in m, j \in n \tag{21}$$

$$SC_{ij} * x_{ij} \leq B_i, \forall i \in m, j \in n \tag{22}$$

$$x_{ij} = 0, \ \textit{if } j \textit{ cannot satisfy deadline of } i, \ \forall\, i \in m;\ j \in n \tag{23}$$

$$x_{ij} = 0, \ \textit{if } j \textit{ cannot satisfy budget of } i, \ \forall\, i \in m;\ j \in n \tag{24}$$

Query Scheduling Times Constraints (25) guarantee the scheduling times of a query by specifying x_{ij} as 1 so that i is guaranteed to be scheduled to one cloud resource for execution in order to satisfy SLAs defined with users.

$$\sum\nolimits_{i \in m, j \in n} x_{ij} = 1 \tag{25}$$

Data Locality Constraints ensure that queries can only execute on resources where datasets can be accessed. The aim is to avoid lengthy big data transfer times and expensive big data transfer costs, as shown in (26). If a resource j has no access to the large datasets to be processed by i, $x_{ij} = 0$ guarantees i is not scheduled to j for SLA guarantees. While $x_{ij} = 1$ is set where the scheduling is required for the optimal scheduling solution.

$$x_{ij} = 0, \ \textit{if } j \textit{ has no access to data of } i, \ \forall\, i \in m, \ j \in n \tag{26}$$

Data Sampling-based Query Resource Constraints, as shown in (27), ensure that the sum of data sampling time to obtain samples and the processing time to execute task on samples using j can be satisfied by available time of j. ST_{ij} represents the required Sampling Time (ST) to sample large dataset and process query i using j.

$$\sum\nolimits_{i \in m, j \in n} \left(ST_{ij} * x_{ij}\right) \leq t_j + T_j \tag{27}$$

Query Accuracy Constraints determine the available resource configurations needed to execute queries without violating query deadlines and budgets. The accuracy of queries determines the data sample size and associated resources to execute the data sample. Increased accuracy requires that a larger data size will be selected and more computing resources will be consumed to process the larger sample. Resource j with a configuration not satisfying the accuracy requirements of queries will be eliminated from the selectable resource pool, as shown in Constraint (28), to enable the ILP solver to obtain the optimized solution in a reduced search space.

$$x_{ij} = 0, \ \textit{if } j \textit{ cannot satisfy accuracy of } i, \ \forall\, i \in m; \ j \in n \tag{28}$$

Data Sampling-based Query Deadline Constraints (29) guarantee if $b_{ik} = 1$, i should finish sampling data and executing query on data sample no later than the start time of query k. Non-linear constraint (32) is used to generates (29) with F_5 satisfies (33) as a sufficient large constant. Constraint (30) ensure if i executes on j at s_i, they should finish sampling and execution no later than D_i. Non-linear constraint (34) is used to generate (30) with F_6 as a constant that is sufficient large satisfies (35). Non-linear relationship (36) is used to generate (31) with F_7 satisfies (37) as a sufficient large constant. This restricts if i executes on j, the data sampling and processing time of i should not exceeds the purchased resource time of j as the sum of EPT_j and t_j.

$$s_i - s_k + F_5 * b_{ik} \leq F_5 - ST_{ij}, \forall i,k \in m; j \in n \tag{29}$$

$$s_i + F_6 * x_{ij} \leq F_6 + D_i - ST_{ij}, \forall i \in m, j \in n \tag{30}$$

$$s_i - t_j + F_7 * x_{ij} \leq F_7 - ST_{ij} + EPT_j, \forall i \in m, j \in n \tag{31}$$

$$b_{ik} = 1 \Rightarrow s_i + ST_{ij} \leq s_k, \forall i,k \in m, j \in n \tag{32}$$

$$F_5 \geq \max\ (s_i + ST_{ij} - s_k) + 1, \forall i,k \in m, j \in n \tag{33}$$

$$s_i * x_{ij} + ST_{ij} \leq D_i, \forall i \in m, j \in n \tag{34}$$

$$F_6 \geq max(s_i + ST_{ij} - D_i) + 1, \forall i \in m, j \in n \tag{35}$$

$$x_{ij} = 1 \Rightarrow s_i + ST_{ij} \leq t_j + EPT_j, \forall i \in m, j \in n \tag{36}$$

$$F_7 \geq max(s_i - t_j + ST_{ij} - EPT_j) + 1, \forall i \in m, j \in n \tag{37}$$

3 Admission Control and Resource Scheduling

Big data analytics faces the challenges of tight budgets and/or deadlines, where queries cannot always be fully admitted whilst satisfying all QoS requirements with SLA guarantees during AaaS delivery. To tackle such research challenges, data sampling-based optimization algorithms are proposed that process data samples and return AaaS solutions in a faster manner with significantly reduced resource costs and enhanced profits.

3.1 Admission Control

Queries are submitted online to the AaaS platform from various domain users. The admission controller iteratively admits each query. The pseudo code of the admission control algorithm is shown in Algorithm 1. The admission controller first checks if BDAA is available and the associated dataset is accessible. If so, it further estimates if

Algorithm 1: Query Admission Control Algorithm

```
Input: big data analytics requests/ queries, BDAAs, cloud resources, resource configurations
Output: query acceptance or rejection decisions
1: for every submitted query ∈ submitted big data analytics requests
2:    requested BDAA by the query ← broker/search the BDAA registry
3:    if the BDAA is brokerable/available && big data can be accessed
4:       configurations ← search the resource registry for all configurations
5:       if configurations > 0
6:          for every configuration ∈ all available resource configurations
7:             ET ← estimate the execution time on the resource configuration
8:             EC ← estimate the query cost on the resource configuration
9:             if EC < budget && ET < deadline
10:               admQueries ← query admission as fullQuery
11:            end if
12:            else if the query request allows sacrificing accuracy for reduced times and costs
13:               apprQueries ←get queries support approximate processing
14:               for every query request in the apprQueries
15:                  SC ← estimate the cloud resource cost of data sampling
16:                  ST ← estimate the required time of data sampling
17:                  DPC ← estimate the cloud resource cost to process data samples
18:                  PT ← estimate the time to process data samples
19:                  QA ← estimate data accuracy to process data samples
20:                  if SC + DPC < budget && ST + PT < deadline && A > accuracy
21:                     admQueries ← query admission as apprQuery
22:                     avaiResources ← update the available cloud resources
23:                  end if
24:               end for
25:            end if
26:       end for
27:       if the query request is not successfully admitted
28:          reject the data analytics request with rejection details
29:       end if
30:    end if
31:end for
```

the QoS requirements of budget, deadline, and accuracy can be satisfied by any existing cloud resource configuration in the AaaS platform or whether it can be brokered from third party resource providers. It searches all resource configurations in the registry of cloud resources. The admission controller calculates the Estimated Cost (EC) and Estimated Time (ET) for each configuration to execute a query satisfying the given QoS requirements. If such configuration is found, the admission controller admits the query as *fullQuery* and adds the query to the admission queue (Lines 1–11).

If the admission controller cannot admit a query with tight budgets and/or deadlines, approximate processing of the query is considered only if the results are meaningful without affecting reliable decision making. The admission controller estimates if QoS requirements can be satisfied through the support of data sampling considering the overall query cost that is calculated as the sum of the Sampling Cost (SC) and the Data Processing Cost (DPC). The overall query time is calculated as the sum of data Sampling Time (ST) and Processing Times (PT). Furthermore, the admission controller estimates whether the query Accuracy (A) can be fulfilled by processing the data samples on given resource configuration (Lines 12–19).

BDAA profiles provisioned by third-party providers contain information of the BDAA costs, the required resource times including sampling times and query times, as well as the resource configurations for different query requests. BDAA application

profiles are the bases for the AaaS platform to make accurate estimation of query processing time, accuracy, and cost for admission and scheduling decisions. BDAA costs for the same BDAA can be different for different versions, e.g. sequential processing versions usually costs less than parallel processing versions.

Based on the application profiles, we can obtain the query processing times for given resource configurations, the accuracy on given resource configurations, and the BDAA costs to decide whether QoS requirements of queries can be fulfilled. This is achieved by using estimation method in the following way: for a given query, based on the accuracy requirement, profiles satisfying accuracy can be preliminarily selected. After considering budget requirements, all of the profiles that are still possible to be selected to execute a given query within user specified budgets are selected. The admission controller further calculates the overall costs including BDAA costs and resource costs for available BDAA profiles to satisfy the budget constraints of queries.

For a given query that supports approximate processing, if all QoS requirements can be satisfied by at least one cloud resource configuration, such a query is acceptable and executable as *apprQuery* and the SLAs are established by the SLA manager; otherwise, the query has to be rejected to avoid significant penalty costs caused by SLA violations. Afterwards, the AaaS platform utilizes optimization algorithm to make scheduling decisions to maximize the profits while minimizing query response times in the AaaS platform. After the data sampling-based admission control, rejection reasons are given to users to subsequently modify query specifications for potential resubmission (Lines 20–31).

Admitted queries can be assigned to existing resources for execution if the current resources have sufficient capacity; otherwise, new cloud resources are created to execute the query following the SLA agreements. The optimal query assignment and resource provision decisions are ultimately provided by the resource scheduler for profit maximization and query time minimization.

3.2 Resource Scheduling

To provide timely, reliable, and cost-effective scheduling solutions for the AaaS platform, a data SAmpling-based Profit Optimization (SAPO) scheduling algorithm is put forward. SAPO offers scheduling with SLA aware and data sampling-based methods to provision resources to query processing on data samples to meet QoS requirements of budget, deadline, and accuracy. The pseudo code of the SAPO resource scheduling algorithm is shown in Algorithm 2.

SAPO allows data analytics results to be returned in a time-efficient and reliable manner with controllable resource costs that can benefit users with limited budgets and with timely decision making requirements where accuracy bounds is necessary for reliable decision making. Resource scheduling is the core function of the AaaS platform required to coordinate all the other components to deliver satisfactory services to users through the SAPO algorithm.

The AaaS service delivery scenario starts when users submit queries to the AaaS platform. Each query requests for specific BDAA to analyze the data. For a BDAA, the SAPO scheduler first obtains information from all queries and resources supporting the

BDAA, along with the up-to-date information on the resource configurations from the AaaS platform as the input to the scheduler.

The SAPO scheduler calls the admission controller to admit query requests. Queries are admitted by the admission controller in the following two scenarios. **Scenario 1**: sufficient budgets as well as deadlines are given by users to execute the queries that can meet SLA guarantees on AaaS delivery based on processing the full datasets, named as *fullQueries*. **Scenario 2**: tight deadlines and/or limited budgets are given by users. In this way, processing the entire dataset is not permitted. If accuracy of queries can be traded-off where effective approximate processing is supported for reliable data analytics results, data sampling is used to approximately process smaller sampled datasets to tackle the time and cost challenges with reliable accuracy bounds presented to users, such queries are named as *apprQueries*. SAPO schedules queries by applying the *SAPOOptimization* method that adopts and implements the formulation of the optimization scheduling problem by applying the ILP programming model with Objective Z subjecting to a range of optimization scheduling constraints (1)–(37).

For queries that cannot be executed in full by processing the entire datasets under tight budgets and/or deadlines, the admission controller attempts to admit *apprQueries*. If the sampling technique can deliver satisfactory AaaS services, such *apprQueries* are admitted to the AaaS platform. The SAPO scheduler then generates *anaQueries* from all admitted analytics tasks. The scheduler then provisions *heuCloudResources* applying the *selectSampledResHeuristic* method. Furthermore, the SAPO scheduler utilizes the proposed *SAPOOptimization* algorithm by applying the ILP programming model to create profit-maximization and cost-minimization scheduling solutions (Lines 1–12).

If the generated optimization solutions are feasible that is returned before system-defined scheduling timeout, the optimal solution is then utilized to guide query execution and resource provisioning in the AaaS platform (Lines 13–16). If no scheduling solution is generated before the timeout setting of the optimization algorithm, a heuristic approach named as *DTHeuristic* is used to generate alternative heuristic solutions to avoid SLA violations caused by the failure of query execution. After the scheduling solution is created for the current schedule, the SAPO scheduler triggers auto-scaling to downgrade the capacity by terminating active cloud resources that are idle to save costs at checkpoints by the *scaleDown* method (Lines 17–23).

The ***SAPOOptimization*** method (Lines 24–30) first obtains the current platform information regarding the queries, resources, and BDAAs that are used as the input to the ILP solver. SAPO defines and implements the optimization objectives and constraints. SAPO enables the objective function to maximize the profits and minimize query responses for the AaaS platform. SAPO is subject to various constraints (1)–(37) based on the mixed ILP formulation of the optimization problem. SAPO support data sampling and SLA aware resource scheduling solutions that are designed to tackle the resource scheduling challenges for fast, reliable, and cost-effective optimization solutions to improve the performance and quality of service delivery by the AaaS platform.

The ***DTHeuristic*** method (Lines 31–33) applies a maximum delay time-based heuristic algorithm to map *admQueries* to *exeCloudResources* and executes queries at the earliest available start time on the selected cloud resources.

Algorithm 2: SAPO Cloud Resource Scheduling Algorithm

Input: user submitted big data analytics requests, cloud configurations, all available BDAAs, cloud resources.
Output: optimized | heuristic resource scheduling solution.
1: **for** each requested BDAA ∈ all available BDAAs
2: admQueries ← admissionControl (queries, resourceConfigurations, BDAA)
3: fullProcessingTasks ← obtain all fullQueries from admitted queries
4: apprQueries ← obtain all approximate queries from admitted queries
5: **for** every query request in apprQueries
6: dataSamplingTasks ← create data sampling tasks
7: **end for**
8: anaQueries ← fullProcessingTasks + dataSamplingTasks
9: **if** (anaQueries > 0)
10: resources ← obtain all existing cloud resources running BDAA
11: heuCloudResources ← **selectSampledResourceHeuristic** (resources, configurations, anaQueries, BDAA)
12: optimalSolution ← **SAPOOptimization** (anaQueries, BDAAs, heuCloudResources)
13: **if** a feasible solution is obtained before timeout setting
14: apply the optimal solution to provision resources for query execution
15: **end if**
16: **else**
17: heuristicSolution ← **DTHeuristic** (heuCloudResources, anaQueries, BDAA)
18: apply the heuristic solution to run queries
19: **end else**
20: activeResources ← update existing active resources running BDAA
21: scaleDown (activeResources)
22: **end if**
23: **end for**
24: **Procedure: SAPOOptimization** (anaQueries, heuCloudResources, BDAA)
25: getUpdateToDateInfo (anaQueries, heuCloudResources, BDAA)
26: defineOptimizationScheduleObjectives (**Objective Z**)
27: defineOptimizationScheduleConstraints (**Constraints (1)-(37**))
28: defineOptimizationTimeOut (schedulingTimeout)
29: solveOptimizationScheduleProblem ()
30: optimalSolution ← getOptimizationScheduleSolution ()
31: **Procedure: DTHeuristic** (admQueries, heuCloudResources, BDAA)
32: sort admQueries based on maximum delay time
33: schedule admQueries to resources with minimized earliest start time
34: **Procedure**: **selectSampledResHeuristic** (resources, BDAAs, configurations, admQueries)
35: avaiCloudResources ← obtain all available cloud resources
36: exeCloudResources ← obtain resources from avaiCloudResources
37: newCloudResources ← generate new resources to execute queries
38: heuCloudResources ← exeCloudResources + newCloudResources

The ***selectSampledResourceHeuristic*** method (Lines 34–38) significantly reduces the Algorithm Running Time (ART) of *SAPOOptimization* by reducing the problem search space to efficiently generate data sampling-based optimized resource scheduling solutions. The method first selects *avaiCloudResources* as the available cloud resources selected from the existing cloud resources. It then selects cloud resources with the capacity to execute at least a *fullQuery* or an *apprQuery* satisfying its QoS requirements utilizing *avaiCloudResources* that is defined as *exeCloudResources*. The s*electResourceHeuristic* further generates new resources to execute the submitted queries if the platform does not have sufficient resource capacity to execute newly submitted query requests. The method finally generates *heuCloudResources* as the

summation of *exeCloudResources* and *newCloudResources* that are then input to the *SAPOOptimization* method. The *heuCloudResources* is able to provision sufficient resources that enable *SAPOOptimization* to find feasible solutions to schedule queries satisfying SLAs. The *heuCloudResources* method provisions sufficient resources with capacity close to the optimal configuration created by *SAPOOptimization*. The aim is to reduce the search space to enable *SAPOOptimization* to return optimal solutions in a timely, reliable, cost-effective way.

We compare the performance of SAPO to a Profit Optimization (PO) scheduling algorithm. PO [3] serves as a suitable comparison algorithm as it serves the same objective as SAPO to maximize the profits for the AaaS platform while providing AaaS with minimized queries times for users. Moreover, PO also builds on ILP-based formulation and provides optimal resource scheduling solutions in the AaaS platform. PO applies SLA-based scheduling mechanisms that are able to deliver optimal solutions subject to different constraints, which are: resource capacities, budget requirements, deadline requirements, task execution times and sequences, task dependencies, and data locality constraints [3]. Since PO is not able to support sampling-based scheduling solutions to process big data under constraints of tight deadlines and budgets, it has the limitations in only admitting and scheduling queries with sufficient budgets and deadlines. Thus, PO is not able to tackle big data challenges incurring expensive costs or lengthy processing times where fast, cost-effective, and reliable AaaS is required by decision making in BDAA domains such as banking and stock market. To face such challenges, SAPO serves as the ideal optimization resource scheduling algorithm that is able to deliver optimal solutions not only support big data analytics scenarios enabled by PO but also support big data analytics within tight deadline and limited budget constraints by applying data sampling-based scheduling for timely, cost-efficient, and reliable AaaS solutions in the cloud computing environments.

4 Performance Evaluation

We conducted experimental evaluations to analyze the efficiency of the profit maximization and cost minimization algorithms including SLA guarantees, query admission control, cost saving, resource configuration, profit enhancement, accuracy analysis, and ART analysis.

Experiment Setup: We built the AaaS framework using Cloudsim [7] and utilize IBM CPlex 5.5 as the optimization ILP solver [8]. We conducted experiments for real time and periodic cloud resource scheduling with different Scheduling Intervals (SIs).

Resource Configuration: 4 datacenters are simulated. Each datacenter consists of 500 nodes while each node contains 400 CPU, 10 PB storage, 30 TB memory, and bandwidth of 10 GB/s. Six types of VMs are considered as memory optimized Amazon EC2 VMs: r4.large, r4.xlarge, r4.2xlarge, r4.4xlarge, r4.8xlarge, and r4.16xlarge [9]. A resource can be a CPU core, a single or a cluster of VMs. The unit for memory, storage, cost, and SI is GiB, GB, dollar, minute accordingly.

Data Analytics Workload utilizes Big Data Benchmark [10] and BlinkDB data sampling workload [11]. The big data benchmark provides query response times and

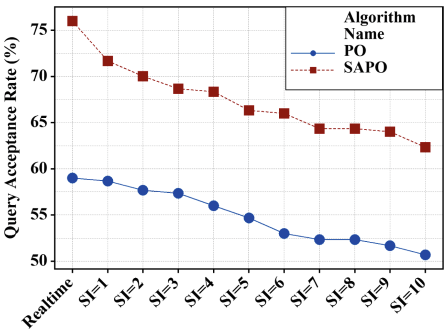

Fig. 1. Admission rates of SAPO and PO.

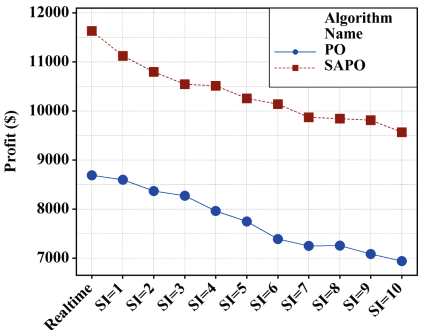

Fig. 2. Profits of SAPO and PO.

the cloud resource configurations using Amazon EC2 VMs to execute on big data analytic frameworks with detailed data size, data location, and data type of datasets. The BlinkDB technique samples the original datasets for sampling processing of the data. The BlinkDB workload provides the resource times, configurations to sample big data with query times and error bounds on accuracies. Based on the big data benchmark and BlinkDB, the times for data sampling and query processing on various configurations using BDAAs are modeled.

Query Information: *Submission time* is generated using one-minute mean Poisson arrival interval. *Query type* has 4 types including scan, join, aggregation, and user defined function. BDAAs has 3 types including Hive on Hadoop as BDAA 1, Hive on Spark without caching as BDAA 2, and Hive on Spark with caching as BDAA 3. *Resource time* contains the resource requirements of queries. Two types of deadline and budgets are considered: tight, which are generated with a Normal Distribution of (3, 1.4), and loose, which are generated with a Normal Distribution of (8, 3). *Accuracy* is generated-based on the BlinkDB sampling workload which details the guaranteed accuracy with given response times and cloud resource costs to execute the approximate queries with error bounds of results. 5 types of accuracy are considered that are 100%, 99%, 95%, 90%, and 85%.

1. Admission Control and SLA Guarantees: To evaluate the algorithm performance of efficient and effective admission control, we conduct experiments for real time and periodic scheduling where SI is in the range of [1, 10]. We compare the algorithm performance of SAPO compared to PO with results shown in Fig. 1. We can see that the SAPO is able to admit more queries for processing for both real time and periodic resource scheduling with an increased query admission rate in the interval of [12%, 17%]. Higher admission rate for processing queries can creates higher profits, increase user satisfactory levels, and enlarge markets by processing more data analytics requests, and hence is highly preferred. Moreover, results also show that all admitted queries by SAPO and PO are processed with SLA guarantees, which proves the effectiveness of the admission control.

2. Profit Enhancement: We evaluate the profit enhancement advantages of SAPO for real time and periodic scheduling scenarios, as shown in Fig. 2. Results show that SAPO creates significantly higher profit than PO for all scheduling scenarios. The

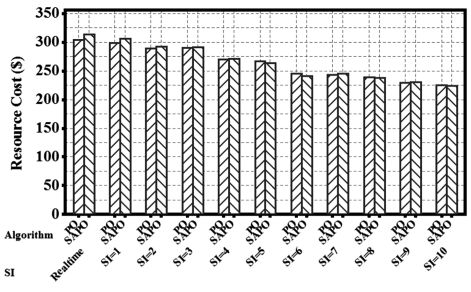

Fig. 3. Resource costs of SAPO and PO.

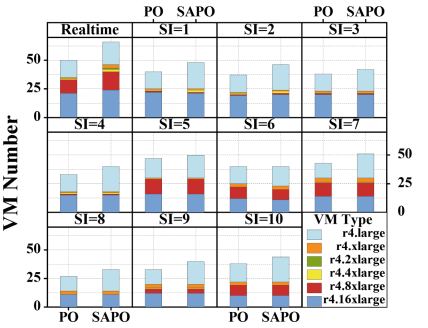

Fig. 4. Resource configurations of SAPO and PO.

increased profit interval is [27.5%, 38.6%]. The enhanced profits come from higher query admission rates and reduced resource consumption from data sampling-based query processing. The profits show a decreasing trend for both SAPO and PO due to the rejection of queries with tight deadlines for periodic SIs. Real time scheduling is not realistic for all online resources since it would generate unnecessarily large computing usage and hence periodic scheduling approaches are supported to schedule queries that arrive during specified SIs. We find SI = 1 is the most suitable periodic SI that can, on the one hand admit more queries, while on the other hand it reduces frequent scheduling computations, which is important for higher admission rates and higher profit enhancements. We further analyze the data processing methods of SAPO and PO and provide details of the data processing methods. We find that the SAPO algorithm supports both full data processing and sampling-based processing of the datasets while the PO scheduling algorithm only supports full data processing. The percentage of queries that are processed using data sampling methods for the SAPO algorithms are in the interval of [16.6%, 22.4%], which shows an increased query rate that benefits creating higher profits for SAPO.

3. Cost Savings: Resource cost is another key performance indicator for algorithm performance. As shown in Fig. 3, we can see that the resource costs of both SAPO and PO are similar. The increased cost rates of SAPO compared to PO are in the interval of [−0.02%, −0.3%]. Under the condition that SAPO admits [12%, 17%] more queries and creates higher profits of [27.5%, 38.6%], such similar resource consumptions and costs clearly indicate the performance advantages of SAPO in cost saving. SAPO is able to reduce costs and increase profits through better optimal decision making. SAPO allows to better utilize resources on larger number of queries with a more comprehensive optimization solution.

4. Resource Configuration: The resource configurations of SAPO and PO for real time and periodic scheduling are shown in Fig. 4. There is a trend whereby r4.large and r4.16xlarge are more frequently utilized by both SAPO and PO. This is consistent with the query workload property whereby the workload is generated with query deadlines and budgets in two categories: tight and loose. For tight deadlines, r4.16xlarge can offer higher processing capacity to meet deadline requirements while r4.large can reduce unit

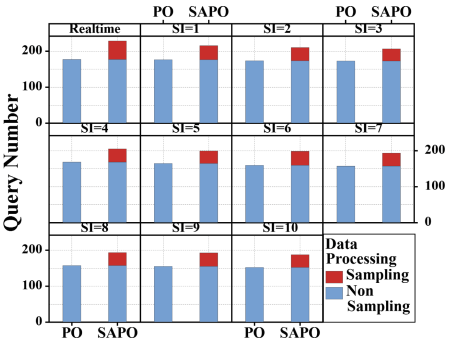

Fig. 5. Query number of data sampling and non sampling processing of SAPO and PO.

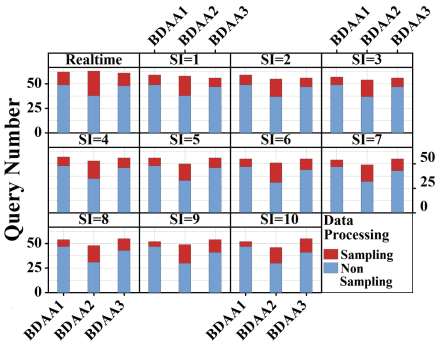

Fig. 6. Query number of sampling and non sampling processing for each BDAA.

resource waste for tight budgets when VMs are idle, hence both r4.large and r4.16xlarge are utilized more frequently. We can also see that the SAPO algorithm has an overall higher number of VMs utilization due to the higher number of admitted queries that require more cloud resources to provide the required computing power for the query workload. We notice the decrease in resource consumption trend for both PO and SAPO due to the decreased query admission rates and hence decreased resource requirements. Decreased resource consumption also comes from the higher number of queries to be scheduled in increased SIs with resultant improvements in resource utilizations.

5. Data Processing Method Analysis: We further analyze the data processing methods of SAPO and PO for better understanding of the performance advantages of SAPO. We obtain the data processing methods of all queries for both SAPO and PO. Results show that the SAPO algorithm supports both full data processing and data sampling-based processing of the datasets while the PO scheduling algorithm only supports full data processing for all BDAAs, as shown in Fig. 5. The percentage of queries that are processed using data sampling methods for the SAPO algorithms are in the interval of [16.6%, 22.4%], which shows that increased query rates create higher profits for SAPO.

After the overall analysis of the algorithm performance for all BDAAs, we further analyze the algorithm performance of SAPO for each BDAA. The number of queries processed by different BDAAs for SAPO and the details of the data processing method are shown in Fig. 6. The number of queries with approximate processing is shown in Fig. 7. We can see that although admitted queries that support approximate processing are in the interval [80.4%, 81.2%] in Fig. 8, SAPO only applies sampling-based scheduling on queries when the QoS requirements cannot be fulfilled to prioritize highly accurate query processing needed for reliable decision making. As a result, approximately processed queries are in the interval [16.5%, 22.4%]. SAPO prioritizes the processing of the entire datasets to provide full accuracy of data analytics results to help users make better strategies and decisions while it deprioritizes the profit gaining

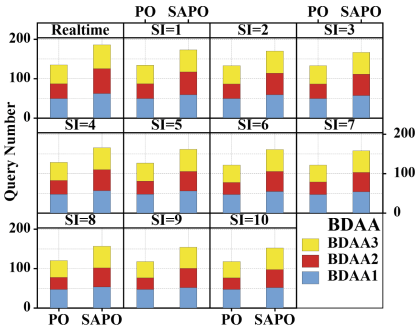

Fig. 7. Approximate query processing number for each BDAA.

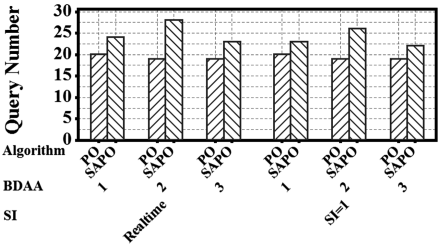

Fig. 8. Query acceptance number for BDAAs.

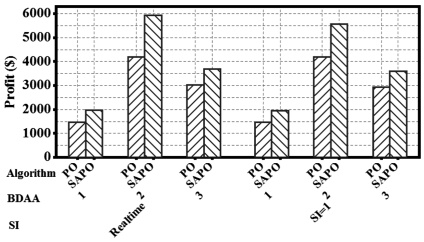

Fig. 9. Profits of SAPO and PO for different BDAAs.

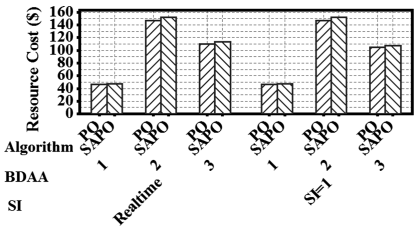

Fig. 10. Resource costs of BDAAs of SAPO and PO for real time and SI = 1

created by approximate processing. SAPO aims to help the AaaS platform to achieve the best performance for higher user satisfaction and enlarge market share by higher quality AaaS delivery.

6. BDAA Analysis: After the algorithm performance analysis for the entire query workload processed by all BDAAs, we further detail the performance evaluation results for each BDAA to show the performance advantages of SAPO. The results show that SAPO outperforms PO for each BDAA with real time scheduling and periodic scheduling using real time scheduling and SI = 1 as examples. The results also show that SAPO is able to admit more queries for both real time and SI = 1 with an increased query acceptance rate in the interval of [15.8%, 42.4%] for BDAA 1-3. The profits created by the SAPO algorithm for BDAA 1-3 are [22.4%, 41.7%] significantly higher than PO as shown in Fig. 9. Moreover, we further analyze the resource costs for SAPO in Fig. 10. With all above performance advantage for query admission and profit enhancement, the resource cost of SAPO is only [0.01%, 0.03%] higher than the PO algorithms for BDAA 1-3, resulting in a significant performance advantage and cost saving. The results also show that SAPO is able to significantly outperform PO for the entire query workload with higher admission rates, higher profits, and lower resource costs. Experimental evaluations also show SAPO significantly outperforms PO for each BDAA for both real time and periodic scheduling.

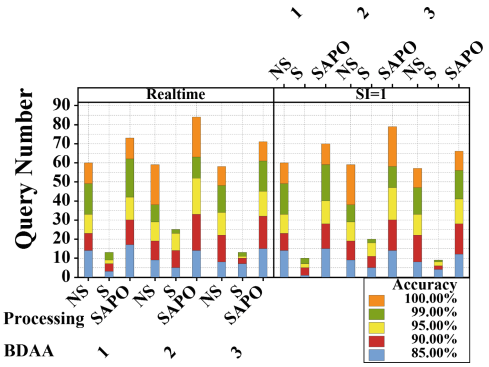

Fig. 11. Accuracy analysis

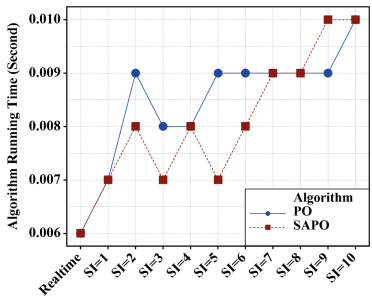

Fig. 12. ART analysis.

7. Accuracy Analysis: We further analyze the accuracy of queries for different query processing methods with results shown in Fig. 11. We can see that for data Sampling (S)-based processing, Non-Sampling (NS)-based processing, and SAPO processing queries with both S and NS-based methods, the number of queries processed with different accuracy requirements varies for different BDAAs. All of the accuracy requirements of the queries are fulfilled with 100% SLA guarantees for both SAPO and PO algorithms. For NS-based processing, we notice that though the query percentage with full accuracy requirements is in the interval of [17%, 36%], NS-based processing method of SAPO does not support approximate processing but only supports full processing of queries. Furthermore, for SAPO, despite [75%, 86%] queries supporting approximate processing, SAPO only process [14%, 30%] of queries with the data sampling-based processing method. The reason is that SAPO prioritizes fully accurate query processing for better decision making though approximate query processing thereby creating higher profits and reducing resource costs. SAPO aims to help the AaaS platform to improve the quality of AaaS delivery for efficient and reliable problem solving and decision making for users in various application domains.

8. ART Analysis: The average ART of SAPO and PO algorithms for real time and periodic scheduling are shown in Fig. 12. The results show an overall increasing trend for ART as more queries are scheduled and more optimization constraints need to be applied for more complex optimization problems with larger inputs and hence a larger solution space. Both SAPO and PO can deliver resource scheduling solutions within 6–10 ms that illustrates the efficiency of the scheduling algorithms. As such, ART does not limit the SAPO scheduling algorithm in delivering effective and efficient resource scheduling solutions. In order to prevent an unexpected large number of queries arriving so that the solution space and input scale exceed the capacity of the optimization algorithms to deliver solutions within SIs, heuristic algorithms are applied as backup solutions. These are able to make near optimal scheduling solutions to guarantee SLAs for the AaaS delivery. The backup heuristics enable the AaaS platform to provide comprehensive scheduling solutions for unexpectedly large query arrivals to mitigate risks and support AaaS delivery with SLA guarantees.

5 Related Work

We focus on providing optimization algorithms to support AaaS platforms to deliver cost-effective, timely, and reliable AaaS for better problem solving in various application domains.

Zhao et al. [12] study an SLA-based admission control and resource scheduling algorithm that offers a cost-effective solution to increase profits for AaaS providers. However, the solution can be sub-optimal due to two-phase resource scheduling. Zhao et al. [3] propose a profit optimization cloud resource scheduling algorithm that is able to maximize profits for AaaS providers with minimized query processing times with SLA guarantees for users, however, this solution cannot tackle big data challenges with limited times and costs. Zhao et al. [13] addresses the big data analytics challenge of resource scheduling under limited deadlines and budgets where accuracy can be traded-off by applying data splitting-based query admission control and profit optimization cloud resource scheduling algorithms. The approach trades-off accuracy for reduced costs and quicker times while guaranteeing budget and deadline requirements of queries, however, it does not provide accuracy guarantees during AaaS delivery that are essential to support reliable and accurate decision making.

There are related works that diverse the BDAAs for the AaaS platforms. Agarwal et al. [11] propose BlinkDB to approximately process large datasets for reliable results with response time error bounds. Zhang et al. [14] build an ND-based solution to split large medical data in clouds providing faster responses with reduced costs and higher flexibility. Tordini et al. [15] process sequencing datasets of complex workflows applying data splitting techniques while maintaining high accuracy.

Mian et al. [16] apply heuristics to provision resources for data analytics workloads with effective resource configuration in a public cloud, however, they sacrifice SLAs to reduce resource costs. This is different from our SLA guarantee purpose as satisfactory AaaS delivery is the fundamental for user satisfaction. Wang et al. [17] support data-aware scheduling for efficient load balancing, however, they do not consider QoS, SLA guarantee, or cost optimization. Xia et al. [18] support a fair and collaborative way to place big data into distributed datacenters, however, they do not consider QoS, SLAs, or indeed AaaS delivery.

Garg et al. [19] manage resources for general cloud applications instead of big data analytics applications. Gu et al. [20] support cost-minimized big data analytics in distributed datacenters, however, does not support cost-optimization and SLA guarantees. Zheng et al. [21] proposed a variety of algorithms to schedule big data workflows on clouds to minimize the costs under deadlines. However, they do not provide automatic scheduling, nor consider budget constraints or consider admission control satisfying SLAs. Dai et al. [22] propose a multi-objective resource allocation approach for big data applications to obtain optimal deployment of VMs in clouds. However, they neither consider admission control nor tackle the time challenges for big data processing.

Zhou et al. [23] propose a declarative optimization engine to provision cloud resources for workflows utilizing available GPU power for timely solutions. They neither consider QoS requirements nor SLA guarantees. Mao et al. [24] propose an

auto-scaling mechanism for traditional workflow scheduling to satisfy task deadlines while minimizing financial costs. However, their approach does not target big data analytics or consider budget or accuracy requirements of requests. Chen et al. [25] propose a utility model-based scheduling mechanism for cloud service provider to optimize profits or user satisfaction. However, they do not consider budget, deadline, accuracy as the key QoS factors of tasks. They trade-off customer satisfaction while we aim to maximize user satisfaction by providing high quality AaaS satisfying query QoS requirements.

The novelty of our research work is proposing data sampling-based admission control and cloud resource scheduling algorithms with SLA guarantees on budget and deadline requirements to support big data analytics solutions under limited times and tight budgets. Moreover, we provide accuracy bounds for big data analytics results to support reliable and accurate decisions. Our proposed algorithms are able to provide profit optimized and time minimized AaaS solutions for fast, cost-effective, and reliable problem solving and decision making in various BDAA domains, which cannot be supported by the above related works.

6 Conclusion and Future Works

Admission control and resource scheduling serve as the key functions to maximize profits and minimize query response times for AaaS platforms to deliver SLA-guaranteed AaaS to various domains of users. We proposed timely, cost-efficient, and reliable query admission control and resource scheduling solutions by modeling, formulating, and implementing the data sampling-based multi-objective optimization solutions. Experimental evaluation shows the admission control and scheduling algorithms significantly increase admission rates, increase profits, and reduce resource costs with efficient resource configurations for the entire the query workloads and for different BDAAs under real time and periodic scheduling scenarios compared to the state-of-the-art algorithms. Furthermore, our proposed algorithms significantly benefit big data analytics under tight deadlines and limited budgets by supporting fully accurate query processing as well as sampling-based query processing that enable users to obtain cost and time effective big data solutions with accuracy guaranteed AaaS for timely, cost-efficient, and reliable decision making.

As part of the future research works, we will continue investigating and proposing automatic and efficient optimization algorithms to tackle big data analytics challenges under various QoS requirements. We will keep working on proposing effective and efficient sampling and splitting-based optimization algorithms to help AaaS platforms to deliver satisfactory AaaS to various domains of users offering faster query response times and reduced cloud resource costs without compromising the reliability of big data analytics solutions.

References

1. Assunção, M.D., Calheiros, R.N., Bianchi, S., Netto, M.A., Buyya, R.: Big data computing and clouds: trends and future directions. J. Parallel Distrib. Comput. **79**, 3–15 (2015)
2. Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A., Khan, U.: The rise of 'big data' on cloud computing: review and open research issues. Inf. Syst. **47**, 98–115 (2014)
3. Zhao, Y., Calheiros, R.N., Bailey, J., Sinnott, R.: SLA-based profit optimization for resource management of big data analytics-as-a-service platforms in cloud computing environments. In: Proceedings of the IEEE International Conference on Big Data, pp. 432–441 (2016)
4. Chaudhuri, S., Das, G., Narasayya, V.: Optimized stratified sampling for approximate query processing. ACM Trans. Database Syst. (TODS) **32**(2), 9 (2007)
5. Benayoun, R., De Montgolfier, J., Tergny, J., Laritchev, O.: Linear programming with multiple objective functions: step method (STEM). Math. Program. **1**, 366–375 (1971)
6. Isermann, H.: Linear lexicographic optimization. OR Spektrum **4**, 223–228 (1982)
7. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**(1), 23–50 (2011)
8. IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/developerworks/downloads/ws/ilogcplex/. Accessed 03 Dec 2018
9. Amazon EC2. http://aws.amazon.com/ec2/instance-types/. Accessed 03 Dec 2018
10. Big Data Benchmark. https://amplab.cs.berkeley.edu/benchmark/. Accessed 12 Dec 2018
11. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: BlinkDB: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM European Conference on Computer Systems, p. 29 (2013)
12. Zhao, Y., Calheiros, R.N., Gange, G., Ramamohanarao, K., Buyya, R.: SLA-based resource scheduling for big data analytics as a service in cloud computing environments. In: Proceedings of the 44th IEEE International Conference on Parallel Processing, pp. 510–519 (2015)
13. Zhao, Y., Calheiros, R.N., Gange, G., Bailey, J., Sinnott, R.: SLA-based profit optimization for resource scheduling of big data analytics-as-a-service in cloud computing environments. IEEE Trans. Cloud Comput. 1–18 (2018)
14. Zhang, H., Zhao, Y., Pang, C., He, J.: Splitting large medical data sets based on normal distribution in cloud environment. IEEE Trans. Cloud Comput. (99), 1 (2015, in press)
15. Tordini, F., Aldinucci, M., Viviani, P., Merelli, I., Lio, P.: Scientific workflows on clouds with heterogeneous and preemptible instances. In: Proceedings of the International Conference on Parallel Computing, pp. 605–614 (2017)
16. Mian, R., Martin, P., Vazquez-Poletti, J.: Provisioning data analytic workloads in a cloud. Future Gener. Comput. Syst. **29**(6), 1452–1458 (2013)
17. Wang, K., Zhou, X., Li, T., Zhao, D., Lang, M., Raicu, I.: Optimizing load balancing and data-locality with data-aware scheduling. In: Proceedings of the 2014 IEEE International Conference on Big Data, pp. 119–128 (2015)
18. Xia, Q., Xu, Z., Liang, W., Zomaya, A.Y.: Collaboration-and fairness-aware big data management in distributed clouds. IEEE Trans. Parallel Distrib. Syst. **27**(7), 1941–1953 (2015)
19. Garg, S.K., Toosi, A.N., Gopalaiyengar, S.K., Buyya, R.: SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter. J. Netw. Comput. Appl. **45**, 108–120 (2014)
20. Gu, L., Zeng, D., Li, P., Guo, S.: Cost minimization for big data processing in geo-distributed data centers. IEEE Trans. Emerg. Top. Comput. **2**(3), 314–323 (2014)

21. Zheng, W., Qin, Y., Bugingo, E., Zhang, D., Chen, J.: Cost optimization for deadline-aware scheduling of big data processing jobs on clouds. Future Gener. Comput. Syst. **82**, 244–255 (2018)
22. Dai, W., Qiu, L., Wu, A., Qiu, M.: Cloud infrastructure resource allocation for big data applications. IEEE Trans. Big Data **4**(3), 313–324 (2018)
23. Zhou, A.C., He, B., Cheng, X., Lau, C.T.: A declarative optimization engine for resource provisioning of scientific workflows in IaaS clouds. In: Proceedings of the 24th International Symposium on High Performance Parallel Distributed Computing, pp. 223–234 (2015)
24. Mao, M., Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seatle, WA, pp. 1–12 (2011)
25. Chen, J., Wang, C., Zhou, B.B., Sun, L., Lee, Y.C., Zomaya, A.Y.: Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In: Proceedings of the 20th International Symposium on High Performance Distributed Computing, pp. 229–238 (2011)