Efficient Data Placement and Replication for QoS-Aware Approximate Query Evaluation of Big Data Analytics

Qiufen Xia, *Member, IEEE*, Zichuan Xu, *Member, IEEE*, Weifa Liang, *Senior Member, IEEE*, Shui Yu, *Senior Member, IEEE*, Song Guo, *Senior Member, IEEE*, and Albert Y. Zomaya, *Fellow, IEEE*

Abstract—Enterprise users at different geographic locations generate large-volume data that is stored at different geographic datacenters. These users may also perform big data analytics on the stored data to identify valuable information in order to make strategic decisions. However, it is well known that performing big data analytics on data in geographical-located datacenters usually is time-consuming and costly. In some delay-sensitive applications, the query result may become useless if answering a query takes too long time. Instead, sometimes users may only be interested in timely approximate rather than exact query results. When such approximate query evaluation is the case, applications must sacrifice timeliness to get more accurate evaluation results or tolerate evaluation result with a guaranteed error bound obtained from analyzing the samples of the data to meet their stringent timeline. In this paper, we study quality-of-service (QoS)-aware data replication and placement for approximate query evaluation of big data analytics in a distributed cloud, where the original (source) data of a query is distributed at different geo-distributed datacenters. We focus on the problems of placing data samples of the source data at some strategic datacenters to meet stringent query delay requirements of users, by exploring a non-trivial trade-off between the cost of query evaluation and the error bound of the evaluation result. We first propose an approximation algorithm with a provable approximation ratio for a single approximate query. We then develop an efficient heuristic algorithm for evaluating a set of approximate queries with the aim to minimize the evaluation cost while meeting the delay requirements of these queries. We finally demonstrate the effectiveness and efficiency of the proposed algorithms through both experimental simulations and implementations in a real test-bed, real datasets are employed. Experimental results show that the proposed algorithms are promising.

Index Terms—Data replication and placement, big data analytics, approximate query evaluation, approximation algorithms, algorithm analysis

1 Introduction

With more and more people adopting cloud services, the volume of generated data about user activities and session logs grows at an exponential rate [10], we refer to such large volume of data as *big data*. It is estimated that at least 2.5 quintillion bytes of data are created per day [20]. Such big data plays a significant role in people's daily lives, especially within enterprises, as analytic results of big data can enable the enterprises to better understand their customers' behaviors, and help them seize best growth opportunities in the competitive global markets. Examples of big data analytics include querying user logs to make advertisement decisions, querying network logs to detect potential network attacks such as denial-of-services (DoS) attacks, etc. Querying big data is time-consuming and costly, e.g., a linear scan of a dataset of PB size (10^{15} bytes) takes days even using a solid state drive with a read speed of 6 GB/s, and takes years if the dataset is of EB size (10^{18} bytes) [13]. However, big data of enterprise users needs to be analyzed in a timely manner, such that the obtained analysis results can produce timely business decisions for quick market response. In addition, as cloud service providers such as Amazon, Microsoft, Google, and Facebook are deploying datacenters globally to provide users ubiquitous access to their cloud services [5], [32], [33], [35], big data generated in such cloud services thus is distributed in different geographically located datacenters. Query evaluation of big data analytics in geo-distributed datacenters therefore faces many challenges, as the evaluation

- *Q. Xia is with the International School of Information Science and Engineering, Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Dalian University of Technology, Dalian, Liaoning 116024, China. E-mail: qiufenxia@dlut.edu.cn.*
- *Z. Xu is with the School of Software, Dalian University of Technology, Dalian, Liaoning 116024, China. E-mail: z.xu@dlut.edu.cn.*
- *W. Liang is with the Research School of Computer Science, Australian National University, Canberra, ACT 2601, Australia. E-mail: wliang@cs.anu.edu.au.*
- *S. Yu is with the School of Software, University of Technology Sydney, Ultimo, NSW 2007, Australia. E-mail: Shui.Yu@uts.edu.au.*
- *S. Guo is with the Department of Computing, Hong Kong Polytechnic University, Hung Hom, Hong Kong. E-mail: song.guo@polyu.edu.hk.*
- *A.Y. Zomaya is with the School of Computer Science, University of Sydney, Camperdown, NSW 2006, Australia. E-mail: albert.zomaya@sydney.edu.au.*

Manuscript received 1 Jan. 2018; revised 8 May 2019; accepted 30 May 2019. Date of publication 6 June 2019; date of current version 8 Nov. 2019. (Corresponding author: Zichuan Xu.)
Recommended for acceptance by O. Rana.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2019.2921337

usually needs large volume of source data from multiple datacenters. The first challenge is how to ensure QoS requirements of users in terms of *access delays* (query delays), given that the query results will be used in timely decision-making applications. The second challenge is how to minimize the cost of query evaluations for big data analytics due to precious datacenter and bandwidth resources. The third challenge is how to maximize the accuracy of query evaluation results, i.e., minimize the error bound of evaluation results within a specified value.

One promising solution to tackle the mentioned challenges is Approximate Query Processing, which evaluates queries based on the sample data of the original source data, and returns an approximate result with its error bound being minimized. By leveraging sampling technologies, the query delay can be significantly reduced [4], as an approximate result with a large error bound usually is obtained from samples with large error bound and smaller volume compared with original dataset. To further reduce the query delay, the samples can be replicated and placed to multiple datacenters by evaluating the query based on the samples close to the query. Although data sampling and replication can improve system performance, it does not imply that more sample replicas will lead to better system performance, since the maintenance of data consistency between the 'master' samples of data and their 'slave' sample copies in the network does incur cost [6]. To maximize the benefit by approximate query processing and sample replications, strategically replicating and placing the samples of each dataset in a distributed cloud is essentially critical to minimize the evaluating cost of approximate queries while meeting the delay requirements of the queries of different users. One fundamental problem thus is how to replicate and where to place the samples with different error bounds to different datacenters in a distributed cloud so that the delay requirements of user queries can be met, the evaluation cost of the queries and the error bound of evaluation results can be minimized. Several studies on data placement have been conducted in the past [1], [7], [15], [31]. However, most of these studies neither considered data replications of the generated big data [31] nor took into account the QoS requirements on user access delays [7], [15], [31], not to mention to explore a non-trivial tradeoff between the query evaluation cost and the result accuracy [1], [7], [15], [31]. In contrast, we study data replication and placement for approximate query evaluations of big data analytics in a distributed cloud, with an aim to minimize the query evaluation cost and the error of evaluation results while meeting user QoS (query delay) requirements. To the best of our knowledge, this is the first time that the QoS-aware data replication and placement problems for approximate query evaluation of big data analytics in distributed clouds are considered. We are also the first to strive for a non-trivial tradeoff between the error bound of query evaluation result and the query evaluation cost, by developing efficient algorithms for this purpose.

The main contributions of this paper are as follows. We first formulate two novel QoS-aware data replication and placement problems for approximate query evaluation of big data analytics in a distributed cloud, with the aim to minimize the evaluation cost of queries and the error bounds of their evaluation results while meeting QoSs of users, where the evaluation cost of a query is the sum of resource consumption costs for the query including the data processing, storage, transmission, and update costs. We then propose efficient algorithms for the two problems with a single approximate query and multiple approximate queries, respectively. Specifically, for the problem with a single approximate query, we devise an approximation algorithm with a guaranteed approximation ratio, through exploring a non-trivial trade-off between the evaluation cost of the query and the error bound of its evaluation result. For the problem with multiple approximate queries, we develop an efficient algorithm for it. We finally evaluate the performance of the proposed algorithms through both experimental simulations and real implementations in a test-bed. The experimental results show that the evaluation cost of queries is significantly reduced, compared to another baseline algorithm.

The remainder of this paper is organized as follows. Section 2 introduces the system model and problem definitions, followed by an approximation algorithm for the problem with a single approximate query in Section 3. A heuristic algorithm for the multiple approximate query evaluation problem is devised in Section 4. The performance evaluation of the proposed algorithms is conducted in Section 5. The related work is presented in Section 6, and conclusions are given in Section 7.

2 Preliminaries

In this section, we first introduce the system model. We then give notations on big data, approximate queries, QoS requirements of users, stratified samples, and the query evaluation cost model. We finally define the problem precisely.

2.1 System Model

We consider a distributed cloud $G = (\mathcal{DC}, E)$, which consists of a set $\mathcal{DC}$ of datacenters located at different geographical locations that are inter-connected by a set E of communication links (or Internet paths). Let $DC_i \in \mathcal{DC}$ be a datacenter and $e_{ij} \in E$ a link between two datacenters $DC_i \in \mathcal{DC}$ and $DC_j \in \mathcal{DC}$. The computing resource of each datacenter DC_i is used to evaluate queries, while its storage resource is used to store data and query results. Denote by $\zeta(DC_i)$ and $\xi(DC_i)$ the amount of available computing resource and the capacity of computing resource in datacenter $DC_i \in \mathcal{DC}$, and let r_c be the amount of computing resource allocated to process one unit data. We do not restrict the capacity of storage resource of datacenters, as the storage resource usually is abundant and inexpensive, compared with the expensive computing resource [31]. The processing, storage of data at datacenters and the transmission of data along network links consume various cloud resources and thus incur costs of the cloud service provider. Denote by $c_p(DC_i)$ and $c_s(DC_i)$ the costs for processing and storing a unit of data at DC_i. Denote by $c_t(i,j)$ and $d_t(i,j)$ the transmission cost and delay on link $e_{ij} \in E$ for transferring a unit of data.

2.2 Big Data, Approximate Queries, and Users' QoS Requirements

We refer to the data generated by a user as the *dataset* of the user, and term the specified datacenter for the data storage by the user as the *home datacenter* of the data. Let $\mathcal{S}$ be the

collection of datasets generated by all users, denote by S_j a dataset in $\mathcal{S}$, where $1 \leq j \leq J$ with J representing the number of datasets in $\mathcal{S}$, i.e., $J = |\mathcal{S}|$. Denote by $DC(S_j)$ the datacenter at which dataset S_j is located.

In addition to generating big data, enterprise users also consume data that is generated by themselves and other partners. These users explore potential business values by issuing queries on the data. For example, a user may issue a query like 'count the number of customers satisfying the following condition... with the lowest possible error bound within one hour'. As timeliness for many queries is more important than the accuracy of their evaluation results, in this paper we study approximate queries that help users get a 'rough picture' of multiple datasets, by delivering approximate results with a certain error bound while meeting the stringent query delay requirements of the users. Denote by $Q = \{q_m \mid 1 \leq m \leq M\}$ the set of approximate queries in the system, where M is the number of distinct queries, and q_m is an approximate query by a user. Each q_m may demand several datasets distributed at different datacenters, and let $\mathcal{S}(q_m)$ be the collection of datasets demanded by q_m.

As we consider approximate query evaluations within stringent delay requirements, we refer to the *delay requirement* of a query as its *QoS requirement*, where the delay experienced by the query is defined as the duration between the time query is issued and the time query result is obtained at its home datacenter. Denote by d_m the maximum tolerable delay of query q_m. d_m represents the maximum delay incurred in replicating samples and transmitting intermediate results of all the datasets in $\mathcal{S}$.

2.3 Stratified Samples and Sample Replication

To accurately and quickly answer each approximate query q_m, a set of samples of each dataset $S_j \in \mathcal{S}$ has been created, following the stratified sampling strategy [9]. A *stratified sample* refers to a sample that is not drawn from the whole dataset in a random way, but separately from a number of disjoint strata of the dataset in order to ensure a more representative sample. Each created stratified sample with sample size $|n_{j,k}|$ is referred to as the *origin sample* of the dataset, the sample can return the query result with an error bound $\epsilon_{j,k}$, where $k \in \mathbb{Z}^+$. Following the theory of stratified sampling [9], the error bound $\epsilon_{j,k}$ of a sample with size $|n_{j,k}|$ is inversely proportional to the square root $\sqrt{|n_{j,k}|}$ of its size $|n_{j,k}|$. Therefore, a stratified sample with a larger error bound usually has a smaller size, requiring less computing resource to process and a shorter delay to deliver the query result. We further assume that the origin stratified samples of each dataset S_j are materialized in advance at the datacenter where S_j is generated.

To meet the delay requirement of each approximate query, some *slave samples* of each origin sample of dataset S_j may be created and placed at the other datacenters in addition to its home datacenter. Data updates will be performed for each slave sample if there is any update on its origin sample to ensure the slave sample consistent with its origin sample. We assume that the average data size of an update operation is $\psi \cdot |n_{j,k}|$, where ψ is a constant with $0 \leq \psi < 1$ [24] and $|n_{j,k}|$ is the sample size. It is obvious that more slave samples are in the system, the QoS requirements of more user queries tend to be satisfied. However, the update and storage costs on these slave samples will subsequently grow, too. Therefore, it is crucial to create and place a proper number of slave samples with certain error bounds for each origin sample in different datacenters.

Evaluating an approximate query q_m is to abstract the intermediate results from the samples of its requested datasets (possibly in different datacenters), and aggregate the intermediate results at the home datacenter of the query. Let $h(q_m)$ be the home datacenter of q_m. Without loss of generality, we assume that the intermediate result on the component sample of each dataset S_j is proportional to the sample volume, i.e., $\beta \cdot |n_{j,k}|$ for sample $S_{j,k}$, where β is a constant with $0 < \beta \leq 1$ [24]. Since the result of query q_m is aggregated from the intermediate results of the placed samples of different datasets, the *average error bound* of the result of q_m is related to the sizes and error bounds of the placed samples. Specifically, assuming that its component sample sizes of the query are $|n_{j_1,k_1}|$, $|n_{j_2,k_2}|$ and $|n_{j_3,k_3}|$ with error bounds ϵ_{j_1,k_1}, ϵ_{j_2,k_2}, and ϵ_{j_3,k_3}, respectively, the average error bound of the approximate query result is $\frac{|n_{j_1,k_1}| \cdot \epsilon_{j_1,k_1} + |n_{j_2,k_2}| \cdot \epsilon_{j_2,k_2} + |n_{j_3,k_3}| \cdot \epsilon_{j_3,k_3}}{|n_{j_1,k_1}| + |n_{j_2,k_2}| + |n_{j_3,k_3}|}$, following existing studies [36].

2.4 Cost Model

The cost of evaluating approximate queries in a distributed cloud consists of the following four component costs. The *storage cost* is for storing both origin samples and slave samples of each dataset in $\mathcal{S}$; the *process cost* is the cost for evaluating the samples requested by approximate queries; the *update cost* refers to the cost of keeping slave samples consistent with their origin samples, the update cost model guarantees that our proposed algorithms can make eventual data consistency [6]; and the *transmission cost* is the cost of data transfer within the network by transferring the intermediate results of each query q_m from the datacenters where the requested samples are evaluated to the home datacenter $h(q_m)$ of q_m, and transferring the updated data from each origin sample to its slave samples. Following many existing studies [30], [33], [35], we assume that the storage, process, update and transmission costs are proportional with the volume of data that the distributed cloud stored, processed, updated and transmitted, respectively.

2.5 Problem Definitions

Problem 1. *Given an approximate query q_m for big data analytics that demands a set $\mathcal{S}(q_m)$ of datasets, each dataset $S_j \in \mathcal{S}(q_m)$ is generated at datacenter $DC(S_j)$, a variety of origin samples with different sample sizes and error bounds for each dataset S_j are stored at datacenter $DC(S_j)$, the intermediate results of evaluating q_m on these requested datasets will be aggregated at the home datacenter of q_m, and the delay requirement of q_m is d_m. Assume that the available computing resource in one datacenter may not be enough to evaluate query q_m, the* QoS-aware data replication and placement problem for a single approximate query of big data analytics *is to create a set of slave samples for each origin sample and place the slave samples at strategic datacenters in G such that both the evaluation cost of query q_m and the average error bound of its evaluation result are minimized, subject to the capacity constraint of computing resource in each datacenter while meeting the specified delay requirement d_m of q_m.*

Problem 2. *Given a set $\mathcal{S}$ of datasets and a set of approximate queries $Q = \{q_m \mid 1 \leq i \leq M\}$ for big data analytics, assume that the computing resource in one datacenter may not be sufficient to evaluate a single approximate query while the total computing capacity of all datacenters in $\mathcal{DC}$ is no less than the total demand of all queries. The* QoS-aware data replication and placement problem for multiple query evaluation of big data analytics *is to create a set of slave samples for each origin sample and place the slave samples at strategic datacenters such that the total evaluation cost of all approximate queries in Q and the average error bound of their evaluation results are minimized, subject to the capacity constraint of each datacenter in $\mathcal{DC}$ while meeting the specified delay requirement d_m of each approximate query $q_m \in Q$.*

3 Approximation Algorithm for Evaluating a Single Approximate Query

We here devise an approximation algorithm with an approximation ratio for the QoS-aware data replication and placement problem of a single approximate query q_m, by exploring a non-trivial trade-off between the evaluation cost of the query and the error bound of its evaluation result.

3.1 Algorithm Overview

Without loss of generality, we assume that there is a *base sample* to represent the sample with the smallest size $|n_b|$ of q_m and the largest error bound, and the samples of each dataset in $\mathcal{S}(q_m)$ of query q_m can be divided by $|n_b|$ [22]. Specifically, to create a set of samples for a dataset, we first create the smallest sample of the dataset by setting its targeted size $|n_b|$ with the largest error bound. To minimize the error bounds of different data samples with different sizes, we then create a sample with size $2 \cdot |n_b|$, and so on. This procedure continues until a sufficient number of samples for each dataset is created.

Having stratified samples with different sizes and error bounds for each query, the sample sizes are divisible by $|n_b|$, the basic idea of the proposed approximation algorithm is to first 'split' each sample into a number of base samples, for example the kth sample of dataset S_j can be split into $\frac{|n_{j,k}|}{|n_b|}$ base samples; and then reduce the QoS-aware data replication and placement problem for a single query q_m to the minimum-cost maximum flow in an auxiliary graph $G' = (V', E')$, by considering n_b as a commodity that needs to be assigned. However, the solution to the latter may not return a feasible solution to the former, since a sample consisting of several base samples may be assigned to different datacenters, while each sample can only be processed in one datacenter. We then refine the obtained solution to make sure that the constituent base samples of each sample are assigned to a single datacenter only.

3.2 Approximation Algorithm

The key to the mentioned reduction is how to make a trade-off between the evaluation cost of the query and the error bound of its evaluation result. Specifically, which sample for each dataset in $\mathcal{S}(q_m)$ should be used to minimize both the evaluation cost and the average evaluation error bound, considering that available computing resource in datacenters of G may not be sufficient for the query evaluation of q_m using samples with smallest error bounds. Another important concern on the reduction is how to construct the auxiliary graph G', such that the selected samples for q_m are processed in datacenters with sufficient computing resource. To this end, the proposed approximation algorithm first selects the samples with appropriate error bounds and then assigns the selected samples to different datacenters with sufficient computing resource.

To select the samples for query q_m such that the accumulative available computing resource in all datacenters is enough to evaluate it, we first calculate the total volume of data that can be processed by the available computing resource, that is, $\frac{\sum_{DC_i \in \mathcal{DC}} \zeta(DC_i)}{r_c}$, where $\zeta(DC_i)$ is the amount of available computing resource in datacenter DC_i, and r_c is the amount of computing resource allocated to process one unit of data. We then calculate the total volume of samples with the smallest errors for datasets in $\mathcal{S}(q_m)$, i.e., $\sum_{S_j \in \mathcal{S}(q_m)} |n_{j,1}|$. If $\frac{\sum_{DC_i \in \mathcal{DC}} \zeta(DC_i)}{r_c}$ is smaller than $\sum_{S_j \in \mathcal{S}(q_m)} |n_{j,1}|$, we scale down the size of each sample by a factor of $\gamma = \frac{\sum_{DC_i \in \mathcal{DC}} \zeta(DC_i)/r_c}{\sum_{S_j \in \mathcal{S}(q_m)} |n_{j,1}|}$. For each dataset S_j, we select the sample with a size that is no greater and most closest to size $|n_{j,1}| \cdot \gamma$.

Given the selected samples of datasets that are demanded by the approximate query q_m, the construction of the auxiliary graph $G'(V', E')$ is described as follows.

We start by constructing the node set V' of G'. Let $n_{j,k}$ be a selected sample of dataset $S_j \in \mathcal{S}(q_m)$. For each sample $n_{j,k}$, we create $\frac{|n_{j,k}|}{|n_b|}$ *virtual samples*, with each virtual sample representing a base sample n_b. Denote by $n'_{j,k,\theta}$ the θth virtual sample of $n_{j,k}$. We then add the virtual samples for datasets in $\mathcal{S}(q_m)$ to V', i.e., $V' \leftarrow V' \cup \{n'_{j,k,\theta} \mid \forall S_j \in \mathcal{S}(q_m), \forall k, m\}$. Similarly, we treat each datacenter $DC_i \in \mathcal{DC}$ as $\lceil \frac{\zeta(DC_i)}{r_c \cdot |n_b|} \rceil$ *virtual datacenters* with each having the amount of computing resource to process a virtual sample, i.e., $r_c \cdot |n_b|$. Let $DC'_{i,\iota}$ be the ιth virtual datacenter of DC_i. We add all of such datacenters into V' by setting $V' \leftarrow V' \cup \{DC'_{i,\iota} \mid \forall DC_i \in \mathcal{DC}, \forall 1 \leq \iota \leq \lceil \frac{\zeta(DC_i)}{r_c \cdot |n_b|} \rceil\}$. For the sake of clear demonstration, we add one virtual source node s_0 and a virtual sink node t_0 to V', i.e., $V' \leftarrow V' \cup \{s_0, t_0\}$.

We then set the edge set for G'. First, the virtual source node s_0 is connected to all virtual samples. That is, there is an edge from s_0 to each virtual sample $n'_{j,k,\theta}$. The cost and capacity of edge $\langle s_0, n'_{j,k,\theta} \rangle$ are set to 0 and infinity, respectively. Second, there is an edge from $n'_{j,k,\theta}$ to a virtual datacenter $DC'_{i,\iota}$, if datacenter DC_i can process the amount $|n_b|$ of data of sample $n_{j,k}$ within the specified delay requirement of approximate query q_m (i.e., the total delay of replicating virtual sample n_b to DC_i and transmitting its intermediate results to home datacenter $h(q_m)$ is within its specified delay requirement d_m). The cost of such an edge $\langle n'_{j,k,\theta}, DC'_{i,\iota} \rangle$ is set to the total cost incurred by (1) transferring the volume $|n_b|$ of virtual sample $n'_{j,k,\theta}$ of dataset S_j from its home datacenter $h(q_m)$ to datacenter DC_i; (2) processing the volume $|n_b|$ of sample in datacenter DC_i; (3) updating the placed virtual sample in DC_i; and (4) storing the volume $|n_b|$ of data in DC_i. The capacity of edge $\langle n'_{j,k,\theta}, DC'_{i,\iota} \rangle$ is set to 1, which means that one virtual sample can be only processed by one virtual datacenter $DC'_{i,\iota}$. Finally, an edge from each virtual

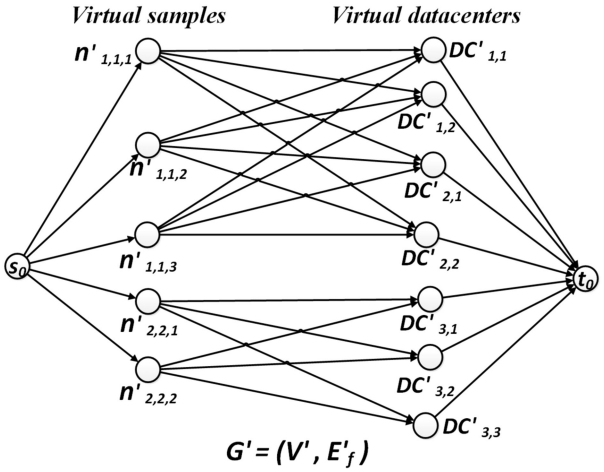


Fig. 1. An example of the constructed auxiliary flow graph $G' = (V', E')$ for the approximation algorithm, where two samples and three datacenters are considered.

datacenter $DC'_{i,t}$ to virtual sink t_0 is added, and its cost and capacity are 0 and infinity, respectively.

Fig. 1 illustrates the constructed auxiliary graph G'. In G', two samples and three datacenters DC_1, DC_2, and DC_3 are considered. The sizes of the samples are $|n_{1,1}|$ and $|n_{2,2}|$, respectively. For the first sample with size $|n_{1,1}|$, three virtual samples $n'_{1,1,1}$, $n'_{1,1,2}$, and $n'_{1,1,3}$ are created, while two virtual samples $n'_{2,2,1}$ and $n'_{2,2,2}$ with size $|n_b|$ are created for the second sample. Similarly, the three datacenters are divided into several virtual datacenters based on their available resource. The delays of replicating the first sample and second sample in datacenters DC_1 and DC_2 and transmitting their intermediate results from DC_1 and DC_2 to its home datacenter are within the delay requirement of the query, so there are edges from each of the samples to the virtual datacenters of DC_1 and DC_2. Notice that there exist key distinctions with the construction of the auxiliary graph in [31]: (1) the construction procedure in [31] did not consider the delay requirement of queries; (2) the constructed auxiliary graph in this paper introduces the concept of 'splitting' each sample to virtual samples and 'dividing' each datacenter to several virtual datacenters, while in the construction procedure in [31] the concept of virtual datacenter is introduced for the sake of moving the capacity constraints to edges.

Having the auxiliary graph G', we now reduce the QoS-aware data replication and placement problem for an approximate query q_m of big data analytics in G to the minimum-cost maximum flow problem in G'. Specifically, we consider each virtual sample as a commodity with demand $|n_b|$ that needs to be routed from s_0 to t_0 in G'. The edge capacity of G' is integral. Following the integrality property for the minimum-cost maximum flow problem [14], there is an integral minimum-cost maximum flow f that routes the commodities (virtual samples) to t_0. That is, the flow from each virtual sample $n'_{j,k,\theta}$ and each virtual datacenter $DC'_{i,t}$ in G' is either 1 or 0, as the capacity of edge $\langle n'_{j,k,\theta}, DC'_{i,t}\rangle$ is 1. This means that each virtual sample will be assigned to a single datacenter. However, such an assignment of virtual samples may not deliver a feasible solution to the original problem, because each sample can only be processed in one datacenter. To deliver a feasible solution, we modify the solution C' obtained from the minimum-cost maximum flow f' in G' to make it become feasible. Let $\{DC_1, \ldots, DC_l, \ldots, DC_L\}$ be the set of datacenters to which the virtual samples of a sample for dataset S_j are assigned in solution C', and denote by DC_{l_0} the datacenter that will incur the minimum cost of processing a virtual sample of S_j in set $\{DC_1, \ldots, DC_l, \ldots, DC_L\}$. We finally assign all virtual samples of $n_{j,k}$ to DC_{l_0}. As a result, the sample of each dataset $S_j \in \mathcal{S}(q_m)$ is assigned to a single datacenter.

The detailed description of the approximation algorithm is given in *Algorithm* 1.

Algorithm 1. `Appro_DPR`

Input: A distributed cloud $G = (\mathcal{DC}, E)$, an approximate query q_m, a set of datasets $\mathcal{S}(q_m)$ with each dataset S_j being generated at datacenter DC_{S_j}, and the delay requirement d_{q_m} of query q_m in accessing some datasets in $\mathcal{S}$.

Output: The placement and replication locations of the samples of datasets in $\mathcal{S}(q_m)$.

1: Calculate the total volume of data that can be processed by the available computing resource, i.e., $\frac{\sum_{DC_i \in \mathcal{DC}} \zeta(DC_i)}{r_c}$;
2: Calculate the total volume of samples with the smallest error bounds for datasets in $\mathcal{S}(q_m)$;
3: **if** $\frac{\sum_{DC_i \in \mathcal{DC}} \zeta(DC_i)}{r_c}$ is smaller than the total volume of samples with the smallest error bounds for datasets in $\mathcal{S}q_m$ **then**
4: Scale down the size of each sample by a factor of $\gamma = \frac{\sum_{DC_i \in \mathcal{DC}} \zeta(DC_i)/r_c}{\sum_{S_j \in \mathcal{S}(q_m)} |n_{j,1}|}$;
5: Construct the auxiliary graph $G'(V', E')$, as shown in Fig. 1, to make sure all the selected samples are assigned to datacenters for processing;
6: Obtain an integral minimum-cost maximum flow f that route the commodities to destination node t_0;
7: Adjust the solution obtained from the minimum-cost maximum flow f' in G', by assigning all virtual samples of each sample to the datacenter DC_{l_0} that is assigned with the virtual sample with the minimum cost;
8: **return** The placement and replications of stratified samples with error bounds of datasets.

3.3 Algorithm Analysis

In the following, we analyze the approximation ratio of the proposed approximation algorithm.

Lemma 1. *Given an approximate query q_m, the samples with specified error bounds of q_m's datasets in $\mathcal{S}(q_m)$, the average error bound of the query result is monotonously increasing with the increase of the error bound of a sample for each dataset $S_j \in \mathcal{S}(q_m)$.*

Proof. Recall that $\mathcal{S}(q_m)$ is the source dataset of query q_m. Consider one dataset $S_1 \in \mathcal{S}(q_m)$ that has two samples with error bounds $\epsilon_{1,1}$ and $\epsilon_{1,2}$ and $\epsilon_{1,1} < \epsilon_{1,2}$. The claim holds if we can show that

$$\frac{|n_{1,1}| \cdot \epsilon_{1,1} + \sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}| \cdot \epsilon_{j,k}}{|n_{1,1}| + \sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}|} < \frac{|n_{1,2}| \cdot \epsilon_{1,2} + \sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}| \cdot \epsilon_{j,k}}{|n_{1,2}| + \sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}|}, \quad (1)$$

where $|n_{1,1}|$ and $|n_{1,2}|$ are the sizes of two samples of dataset S_1, $|n_{j,k}|$ and $\epsilon_{j,k} == \frac{1}{\sqrt{|n_{j,k}|}}$ are the size and error bound of the kth sample of dataset S_j. Specifically, we have

$$\begin{aligned}
&\frac{|n_{1,1}| \cdot \epsilon_{1,1} + \sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}| \cdot \epsilon_{j,k}}{|n_{1,1}| + \sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}|} \\
&= \frac{|n_{1,1}| \cdot \frac{1}{\sqrt{|n_{1,1}|}} + \sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}| \cdot \frac{1}{\sqrt{|n_{j,k}|}}}{|n_{1,1}| + \sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}|}, \\
&= \frac{\sqrt{|n_{1,1}|} + \sum_{j=2}^{|\mathcal{S}(q_m)|} \sqrt{|n_{j,k}|}}{n_{1,1} + \sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}|}.
\end{aligned} \tag{2}$$

Let's take $n_{1,1}$ as a variable x of the function in (2), i.e., $f(x) = \frac{\sqrt{x}+\sum_{j=2}^{|\mathcal{S}(q_m)|} \sqrt{|n_{j,k}|}}{x+\sum_{j=2}^{|\mathcal{S}(q_m)|} |n_{j,k}|}$. Here, x is the size of a sample, and $f(x)$ represents the average error bound of a query evaluated on samples including the sample with size x. Clearly, $f(x)$ is monotonously increasing with the decrease of x ($|n_{1,1}|$), because x decreases faster than $\sqrt{x}$. As the error bound of a sample is inversely proportional to the square root of its size, a sample with a smaller size thus has a larger error bound, when x decreases, the error bound of a query evaluated on x's corresponding sample increases, the average error bound of the query evaluated on samples including the sample increases too. Therefore, the average error bound of the query result is monotonously increasing with the increase of the error bound of a sample for each dataset $S_j \in \mathcal{S}(q_m)$. □

Lemma 2. *Given the selected samples of each dataset $S_j \in \mathcal{S}(q_m)$, all their virtual samples can be routed to the virtual sink t_0 in auxiliary graph G', with the computing capacity of each datacenter being violated by at most a factor of $\frac{|n_b|}{C_{min}}$, where C_{min} is the amount of computing resource of datacenter that has the minimum computing capacity.*

Proof. We first show that the total available computing resource of all datacenters in $\mathcal{DC}$ is enough to evaluate an approximate query q_m based on the selected samples of its datasets in $\mathcal{S}(q_m)$. Recall that we create samples with the smaller sizes (the larger error bounds), if the total computing resource of all datacenters is not enough to process the samples with the largest size (the smallest error bound), then we scale down the sample with the largest size by a factor of $\gamma = \frac{\sum_{DC_i \in \mathcal{DC}} \zeta(DC_i)/r_c}{\sum_{S_j \in \mathcal{S}(q_m)} |n_{j,1}|}$, where $n_{j,1}$ is the sample with the smallest size of dataset S_j. For each dataset S_j, we select the sample with a size that is smaller (or equal) and the most closest to $|n_{j,1}| \cdot \gamma$. This means that the error bounds for processing samples by the available computing resource can be as small as possible.

We then show that the computing capacity of each datacenter can be violated by no more than a factor of $\frac{|n_b|}{C_{min}}$. Given the samples with error bounds specified by users, the samples are divided into virtual samples and the datacenters are treated as virtual datacenters. Each virtual datacenter will be able to process one virtual sample. Recall that each datacenter is virtually treated as $\lceil \frac{\zeta(DC_i)}{r_c \cdot |n_b|} \rceil$ virtual datacenters. If $\zeta(DC_i)$ cannot be divided by $r_c \cdot |n_b|$ exactly, there will be a virtual datacenter with less than the amount $r_c \cdot |n_b|$ of computing resource. The base sample that is assigned to such a virtual datacenter cannot be processed due to resource violation. Therefore, there must be at most one virtual datacenter that does not have enough computing resource to process the volume $|n_b|$ of data. Thus, the computing capacity of a datacenter can be violated by at most $\frac{|n_b|}{C_{min}}$. □

Theorem 1. *Given a distributed cloud $G = (\mathcal{DC}, E)$, an approximate query q_m, and a set $\mathcal{S}(q_m)$ of datasets with each dataset $S_j \in \mathcal{S}(q_m)$ storing at datacenter DC_{S_j}, the tolerable delay requirement d_m, there is an approximation algorithm `Appro_DPR`, for the QoS-aware data replication and placement problem for a single approximate query, which delivers a feasible solution with the approximation ratio of δ in terms of the evaluation cost and the approximation ratio of $\frac{1}{\sqrt{|n_{min}|/|n_{max}|}}$ for the average error bound of the evaluation result, while the delay requirement of each query q_m is met, where δ represents the largest number of virtual samples that can be created from one sample, $|n_{max}|$ and $|n_{min}|$ are the maximum and minimum volume of samples of the datasets in $\mathcal{S}(q_m)$.*

Proof. We first show the approximation ratio in terms of the evaluation cost. Recall that we reduce the original problem to the minimum-cost maximum flow problem, by dividing each sample with the selected error bound into several virtual samples and each datacenter into several virtual datacenters. Let C' be the cost of evaluating a query q_m based on the virtual samples in virtual datacenters. C' is the optimal solution to the problem that allows splitting a sample to multiple datacenters, which can be obtained in polynomial time following the integrality property of flow, since the capacity of edge $\langle n'_{j,k,\theta}, DC'_{i,\iota} \rangle$ is set to either 1 or 0. The delivered solution however is not a feasible solution to the original problem, as the virtual samples that belong to a same sample may be assigned to virtual datacenters of different datacenters.

Let C be the evaluation cost of the proposed approximation algorithm `Appro_DPR`, and C^* be the optimal evaluation cost of the problem. Clearly, C is the cost after adjusting solution C'. Specifically, let $\{DC_1, \ldots, DC_l, \ldots, DC_L\}$ be the set of datacenters to which the virtual samples of a sample for dataset S_j are assigned in solution C', and denote by DC_{l_0} the datacenter that will incur the minimum cost of processing a virtual sample of S_j in set $\{DC_1, \ldots, DC_l, \ldots, DC_L\}$. For simplicity, let $c'_{l,m}$ and $c'_{l_0,m}$ be the costs of assigning a virtual sample to a datacenter in $\{DC_1, \ldots, DC_l, \ldots, DC_L\}$ and DC_{l_0} in solution C', respectively. It is obvious $c'_{l_0,m} \leq c'_{l,m}$. To get the feasible solution C, all virtual samples of S_j that are not in DC_{l_0} are re-assigned to DC_{l_0} following the algorithm `Appro_DPR`. Denote by $c_{l,m}$ the implementation cost of a virtual sample after its re-assignment. Clearly, $c_{l,m}$ is the same as that of the virtual sample assigned to DC_{l_0}. Since $c'_{l_0,m} \leq c'_{l,m}$ in solution C', we have $c_{l,m} \leq c'_{l,m}$ for the re-assigned virtual sample. We then have

$$
\begin{aligned}
C &= \sum_{S_j \in \mathcal{S}(q_m)} c_{S_j} = \sum_{S_j \in \mathcal{S}(q_m)} \sum_{m=1}^{|n_{j,k}|/|n_b|} c_{l_0,m} \\
&\leq \frac{|n_{j,k}|}{|n_b|} \sum_{S_j \in \mathcal{S}(q_m)} \sum_{m=1}^{|n_{j,k}|/|n_b|} c_{l_0,m}, \quad \text{since } \frac{|n_{j,k}|}{|n_b|} \geq 1 \qquad (3)\\
&\leq \frac{|n_{j,k}|}{|n_b|} \sum_{S_j \in \mathcal{S}(q_m)} \sum_{m=1}^{|n_{j,k}|/|n_b|} c'_{l,m}, \text{ since } c_{l_0,m} \leq c'_{l,m} \qquad (4)\\
&= \frac{|n_{j,k}|}{|n_b|} C', \qquad (5)
\end{aligned}
$$

where c_{S_j} is the cost for dealing with samples of dataset S_j.

Since the problem that allows splitting a sample and places split samples to multiple datacenters is a special case of the QoS-aware data placement and replication problem, we have $C' \leq C^*$. By inequality (5), we thus have

$$C \leq \frac{|n_{j,k}|}{|n_b|} C^* \leq \delta \cdot C^*. \qquad (6)$$

The approximation ratio $\frac{C}{C^*}$ thus is $\frac{|n_{j,k}|}{|n_b|}$, where δ represents the maximum number of virtual samples that can be created for any single sample.

We then analyze the approximation ratio for the average error bound of the approximation solution. Let ϵ^* be the optimal average error bound of the evaluation result of q_m. It can be seen that if the total available resource of datacenters can process the samples with the smallest error bound with largest volumes, the optimal average error bound will be the smallest error bound of the samples, i.e., ϵ_1. Thus, we have $\epsilon^* \geq \epsilon_1$. Otherwise, when the total available computing resource is not enough to process the samples with the smallest error bounds, the sample sizes will be scaled down by a factor of γ with $0 < \gamma < 1$. Denote by $\epsilon'_{j,k}$ the error bound of the scaled down sample for dataset S_j. Considering that the error bounds of samples have been determined before hand, there may not be samples $n'_{j,k}$ with an error bound $\epsilon'_{j,k}$; we however can assume that there are samples $n_{j,k}$ with a larger error bound $\epsilon_{j,k}$, it is obvious $\epsilon'_{j,k} \leq \epsilon_{j,k}$ and $|n'_{j,k}| > |n_{j,k}|$. Then, the error bound ϵ achieved by the approximation algorithm `Appro_DPR` is

$$
\begin{aligned}
\epsilon &= \frac{\sum_{S_j \in \mathcal{S}(q_m)} |n_{j,k}| \cdot \epsilon_{j,k}}{\sum_{S_j \in \mathcal{S}(q_m)} |n_{j,k}|} \\
&\leq \frac{\sum_{S_j \in \mathcal{S}(q_m)} |n'_{j,k}| \cdot \epsilon'_{j,k}}{\sum_{S_j \in \mathcal{S}(q_m)} |n'_{j,k}|}, \quad \text{due to Lemma 1} \\
&= \frac{\sum_{S_j \in \mathcal{S}(q_m)} \sqrt{|n'_{j,k}|}}{\sum_{S_j \in \mathcal{S}(q_m)} |n'_{j,k}|} = \frac{\sum_{S_j \in \mathcal{S}(q_m)} \sqrt{|n_{j,1}| \cdot \gamma}}{\sum_{S_j \in \mathcal{S}(q_m)} |n_{j,1}| \cdot \gamma} \\
&= \frac{\sum_{S_j \in \mathcal{S}(q_m)} \sqrt{|n_{j,1}|}}{\sqrt{\gamma} \cdot \sum_{S_j \in \mathcal{S}(q_m)} |n_{j,1}|} = \frac{1}{\sqrt{\gamma}} \cdot \epsilon_1 \leq \frac{1}{\sqrt{\gamma}} \cdot \epsilon^*.
\end{aligned} \qquad (7)
$$

Therefore, the approximation ratio in terms of the average error bound $\frac{\epsilon}{\epsilon^*}$ is no more than $\frac{1}{\sqrt{\gamma}}$. Considering that we assume that although the available computing resource of all datacenters may not be able to process samples with smallest errors with largest volumes, they are enough to process the samples with the largest error bounds. The scale down factor γ thus is no more than $\frac{|n_{max}|}{|n_{min}|}$. The approximation ratio thus is no more than $\frac{\alpha}{\sqrt{|n_{min}|/|n_{max}|}}$. □

4 A Heuristic Algorithm for Multiple Approximate Query Evaluation

In this section, we devise a fast and scalable heuristic algorithm for the QoS-aware data replication and placement problem for multiple approximate query evaluation. We start with a brief overview of the proposed algorithm. We then detail the algorithm. We finally analyze the time complexity of the proposed algorithm.

4.1 Algorithm Overview

A simple solution to the QoS-aware data replication and placement problem with multiple approximate queries for big data analytics is to invoke the proposed approximation algorithm for each of the queries one by one. This solution however does not consider the fact that a dataset can be shared by different queries. Consequently this will create many redundant samples for a dataset than necessary, thereby increasing the storage, transmission and update costs of samples. For example, if all queries request a same dataset, the solution will create a sample for every query, which however can be significantly improved if it only creates one sample for all queries, since all the queries share the same dataset. As different queries not only have different datasets but also have different delay requirements, it is necessary to create different numbers of slave samples for each origin sample at different datacenters to meet their delay requirements. Therefore, an important question is how many slave samples of each origin sample requested by a query should be created, and at which datacenters these slave samples will be placed, To answer this question, we consider not only the locations of existing slave samples of the dataset that are placed by the evaluation of previously considered queries, but also potential locations for newly created slave samples of the dataset with the objective to minimize the evaluation cost of queries that share this dataset. Specifically, we reduce the problem to an unsplittable minimum-cost multi-commodity flow problem in an auxiliary directed graph $G_f = (V_f, E_f; u, c)$ which will be constructed later, where the pair of a query q_m and each of its required datasets in $\mathcal{S}(q_m)$ is referred to as a *commodity*. Each commodity needs to be routed to a datacenter, which corresponds to creating a slave sample for the required dataset at the datacenter, or using an existing slave sample at the datacenter already.

Another important issue is how to determine based on which sample of each dataset in $S_j \in \mathcal{S}(q_m)$ that each query q_m should be evaluated to decrease the error bound of evaluation result and reduce its evaluation cost, considering that a dataset has several different original samples with different sizes and error bounds. To minimize the error bound of the evaluation result for q_m, we consider the sample of dataset S_j with the smallest error bound initially. If the computing resource of all datacenters is not enough to evaluate all

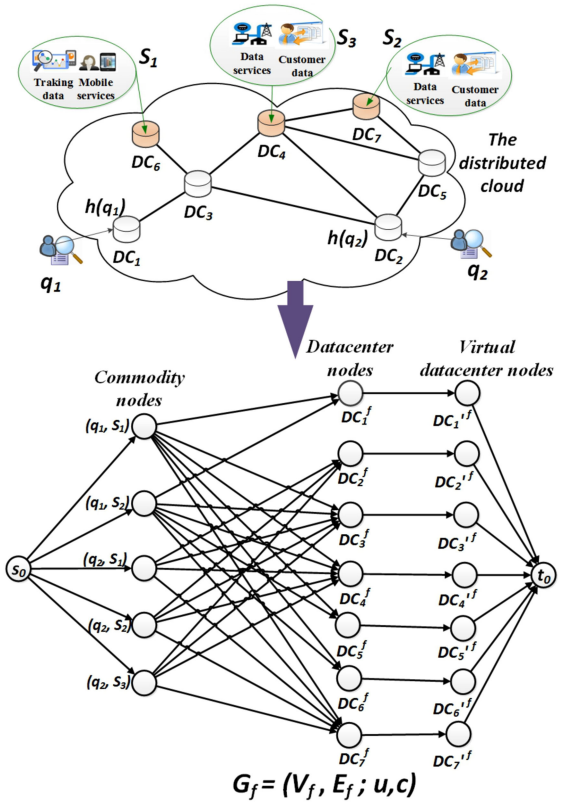


Fig. 2. An example of the distributed cloud G and its auxiliary flow graph G_f, where query q_1 requires the samples of datasets S_1 and S_2 for its evaluation, while query q_2 requires the samples of datasets S_1, S_2, and S_3 for its evaluation.

queries at their smallest error bounds, the algorithm will greedily select some queries by increasing their samples' error bounds, as larger error bounds indicate smaller sample sizes, leading to less computing resource to process.

The mentioned procedure of routing commodities continues until all commodities are successfully routed to their destinations.

4.2 Algorithm

The detailed algorithm proceeds iteratively. Within each iteration, it consists of two phases: (1) the construction of the flow graph $G_f = (V_f, E_f; u, c)$; and (2) routing samples to datacenters for query evaluation. If not all demands can be met by the system, the sample size is reduced and the error bounds are increased. This iteration continues until there is sufficient computing resource for the query evaluation.

The construction of $G_f = (V_f, E_f; u, c)$ is as follows. Since a dataset may be requested by multiple datasets at the same time and their samples can be placed to different datacenters as long as the evaluation cost is reduced, we use a *commodity node* in G_f to represent a query q_m and each of its demanded datasets, i.e., (q_m, S_j), where $S_j \in \mathcal{S}(q_m)$ is a dataset that query q_m demands. Routing this commodity thus means both the assignment of query q_m and the placement of the sample of its required dataset S_j. In other words, to evaluate query q_m, we need to route all commodities of the query to some datacenters in G for processing. Therefore, each datacenter $DC_i \in \mathcal{DC}$ is treated as a *datacenter node* DC_i^f in G_f. Denote by $\mathcal{DC}^f$ the set of datacenter nodes. Each datacenter node $DC_i^f \in \mathcal{DC}^f$ has a *virtual datacenter node* $DC_i'^f$, and a directed edge $\langle DC_i^f, DC_i'^f \rangle$ from DC_i^f to $DC_i'^f$ is added into E_f with the capacity representing the volume of data that can be processed by DC_i, denote by $\mathcal{DC}'^f$ the set of virtual datacenter nodes. In addition, a *virtual source node* s_0 and a *virtual sink node* t_0 are added to G_f, i.e., $V_f = \{s_0\} \cup \big(\bigcup \{(q_m, S_j)\} \big) \cup \mathcal{DC}^f \cup \mathcal{DC}'^f \cup \{t_0\}$, where $S_j \in \mathcal{S}(q_m)$ and $1 \leq m \leq M$.

There is a directed edge from the virtual source node s_0 to each commodity node (q_m, S_j), its capacity and cost are the volume of S_j and 0, respectively. A directed edge $\langle (q_m, S_j), DC_i^f \rangle$ from a commodity node (q_m, S_j) to a datacenter node DC_i^f is added to E_f, if the delay requirement of query q_m for evaluating on a sample at datacenter DC_i is satisfied. The capacity of edge $\langle (q_m, S_j), DC_i^f \rangle$ is the volume of S_j. If a slave sample of S_j has not been placed at DC_i, the cost of edge $\langle (q_m, S_j), DC_i^f \rangle$ is the sum of the storage cost for storing a unit of data at datacenter DC_i, the update cost for updating a unit of data along the shortest path with minimum cost between the datacenter where its origin sample is and DC_i, and the transmission cost for transmitting a unit of intermediate data along the path with minimum transmission cost from DC_i to the home datacenter $h(q_m)$ of q_m. Otherwise, the cost of $\langle (q_m, S_j), DC_i^f \rangle$ is set to the transmission cost for transmitting a unit of intermediate data along the path with minimum transmission cost from DC_i to $h(q_m)$, because the update and storage costs of S_j have been considered when routing other commodities. A directed edge $\langle DC_i^f, DC_i'^f \rangle$ from a datacenter node DC_i^f to a virtual datacenter node $DC_i'^f$ is added to E_f, and its capacity is the amount of data that can be processed by the available computing resource of datacenter DC_i, and its cost is the cost of processing a unit of data at DC_i. Similarly, there is a directed edge $\langle DC_i'^f, t_0 \rangle$ from each virtual datacenter node $DC_i'^f$ to the virtual sink node t_0, its capacity is set to infinity, and its cost is 0.

An example of a constructed graph $G_f = (V_f, E_f; u, c)$ from the distributed cloud $G = (\mathcal{DC}, E)$ is illustrated in Fig. 2, where two approximate queries q_1 and q_2 are issued. The evaluation of query q_1 requires samples of datasets S_1 and S_2, while the evaluation of query q_2 requires samples of datasets S_1, S_2 and S_3. The origin samples of datasets S_1, S_2 and S_3 are located at datacenters DC_6, DC_7 and DC_4, respectively. The delay requirement of q_1 for evaluating on the samples of datasets S_1 and S_2 at datacenter DC_2 cannot be met, so there are no edges between the commodity nodes of q_1 (i.e., (q_1, S_1) and (q_1, S_2)) and the datacenter node DC_2^f. Similarly, there are no edges between the commodity nodes of q_2 (i.e., (q_2, S_1), (q_2, S_2) and (q_2, S_3)) and datacenter nodes DC_1^f, DC_5^f, and DC_6^f, as the delay requirement of q_2 for evaluating on the samples of the datasets located at datacenters DC_1, DC_5, and DC_6 cannot be satisfied.

Having the auxiliary flow graph $G_f = (V_f, E_f; u, c)$, we then route all commodities in G_f one by one. As each dataset requested by a query has multiple samples with different error bounds, we start routing each commodity node by routing its slave sample with the smallest error bound (largest sample size), and iteratively find the sample with next smallest error bound, while each further iteration increases the error bound of the previous iteration until the QoS requirement is met.

Assume that at the beginning of the kth iteration, some queries have been admitted at error bounds from ϵ_1 to ϵ_{k-1}.

We now show how to admit the rest of queries by routing the residual commodities in G_f in the kth iteration. We first route each unrouted commodity (q_m, S_j) by assuming that q_m will be evaluated on the sample of S_j with an error bound ϵ_k. We then rank each unrouted commodity (q_m, S_j) according to the size $|n_{j,k}|$ of the sample of S_j with the error bound ϵ_k, i.e., the rank of query q_m is $\sum_{S_j \in \mathcal{S}(q_m)} |n_{j,k}|$. We further route the commodity (q_m, S_j) with the lowest rank to the destination node t_0 through a path with minimum cost in G_f. If a selected routing path passes a datacenter node DC_i^f, we then create a slave sample for the sample at datacenter DC_i and update the capacities of edges in G_f. Since there is at most one salve sample of a dataset at the same datacenter DC_i, the cost of each edge in set $\{\langle (q'_m, S_j), DC_i^f \rangle \mid q'_m \in Q \setminus \{q_m\}\}$ is updated to the transmission cost by q'_m.

Notice that some queries may not be admitted after K iterations. To admit all queries, we will increase the error bound of samples requested by admitted queries. The procedure of increasing the error bounds of admitted queries continues until all queries are admitted. The detailed procedure of the heuristic algorithm is given in `Algorithm` 2.

Algorithm 2. `Heu_DPR`

Input: A distributed cloud $G = (\mathcal{DC}, E)$, the set of approximate queries Q with each approximate query q_m, a set of datasets $\mathcal{S}$ with each dataset being generated at datacenter $DC(S_j)$, and the delay requirement d_m of each query q_m by evaluating on some datasets in $\mathcal{S}$.

Output: The placement and replication locations of the slave samples of datasets in $\mathcal{S}$.

1: Construct an auxiliary flow graph $G_f = (V_f, E_f)$;
2: Rank the queries in Q according to the volume of their accessed samples in a ascending order;
3: **for** Each query q_m with the minimum rank **do**
4: Get the available computing resource of each datacenter;
5: **for** Each origin sample node S_j accessed by query q_m **do**
6: Treat the origin sample as an unsplittable commodity;
7: Find a path p_{m,j,t_0} from the origin sample node to t_0 with minimum accumulated cost of all edges along the path;
8: Check whether the computing capacity of the datacenter is violated if routing the commodity to the datacenter along path p_{m,j,t_0};
9: **if** The capacity is not violated **then**
10: Route the commodity along the path p_{m,j,t_0};
11: Update the capacities of edges in G_f; Update the cost of each edge in set $\{\langle (q'_m, S_j), DC_i^f \rangle \mid q'_m \in Q \setminus \{q_m\}\}$ as the transmission cost by q'_m;
12: **else**
13: Select the sample with a higher error bound as the commodity to be routed;
14: Repeat step 8 to step 11;
15: **return** The number and locations of samples with different error bounds of all datasets.

4.3 Algorithm Analysis

In the following, we show the correctness and analyze the time complexity of the proposed algorithm `Algorithm` 2.

Theorem 2. *Given a distributed cloud $G = (\mathcal{DC}, E)$, a set Q of approximate queries with the delay requirement d_m for each query $q_m \in Q$, there is an algorithm, `Heu_DPR`, for the QoS-aware data replication and placement problem for multiple query evaluation, which delivers a feasible solution in $O((|Q| \cdot |\mathcal{S}| + |\mathcal{DC}|)^3)$ time.*

Proof. Let p be a flow routing path in G_f starting from s_0 and ending at t_0, i.e., $\langle s_0, (q_m, S_j), DC_i^f, DC_i'^f, t_0 \rangle$. Clearly, the delay requirement of q_m can be met as there will not be an edge from (q_m, S_j) to DC_i^f if the delay requirement can not be satisfied. We then prove that flow f along p corresponds to placing a slave sample of S_j at datacenter DC_i and assigning query q_m to DC_i for evaluating on the placed slave sample. We consider two cases: (1) a slave sample of S_j has already been placed at DC_i; and (2) there is no any slave sample of S_j at DC_i. For case (1), since there is at most one slave sample with an error bound of a dataset at a datacenter, there is no need to create another slave sample for S_j at DC_i. In this way, no extra storage and update costs are incurred, and only processing and transmission costs are incurred. As shown in Step 11 of `Algorithm` 2, the cost of edge $\langle (q_m, S_j), DC_i^f \rangle$ is updated to the transmission cost by query q_m for transmitting a unit of data from its home datacenter $h(q_m)$ to DC_i, and the cost of edge $\langle DC_i^f, DC_i'^f \rangle$ is the process cost of query q_m by processing the placed sample of S_j. Similarly, for case (2) we can show that the flow f via path p corresponds to the placement of a sample of S_j and the assignment of query q_m to DC_i for evaluating on the placed sample.

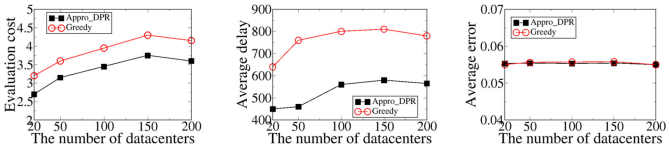
G_f contains $O(|Q| \cdot |\mathcal{S}| + |\mathcal{DC}|)$ nodes and $O(|Q| \cdot |\mathcal{S}| \cdot |\mathcal{DC}|)$ edges, its construction thus takes $O(|Q| \cdot |\mathcal{S}| \cdot |\mathcal{DC}|)$ time. It takes $O(|V_f|^2)$ time to find the shortest path in G_f for each pair of q_m and one of its required dataset from its commodity node $\langle q_m, S_j \rangle$ to t_0, where $|V_f| = O(|Q| \cdot |\mathcal{S}| + |\mathcal{DC}|)$. Thus, the algorithm takes $O(|Q| \cdot |\mathcal{S}| \cdot (|Q| \cdot |\mathcal{S}| + |\mathcal{DC}|)^2)$ time. □

5 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms, and investigate the impact of important parameters on the algorithmic performance, by both simulations and a proof-of-concept in a real test-bed.

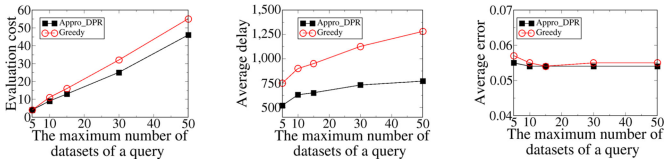
5.1 Experimental Environment

For the simulation, we consider a distributed cloud consisting of 20 datacenters, there is an edge between each pair of datacenters with a probability of 0.2, generated by the GT-ITM tool [11]. The computing capacity of each datacenter is randomly drawn from a value interval [100, 2,000] units (GHz) [31]. Each user produces several Gigabytes of data, we thus emulate the volume of the dataset generated by each user is in the range of [5, 10] GB [31], and the amount of computing resource assigned to the processing of 1 GB data is a value in the range of [20, 50] GHz. The costs of transmitting, storing and processing 1 GB of data are set within a value [$0.02, $0.09], [$0.0275, $0.03], and [$0.05, $0.1], respectively, following typical charges in Amazon EC2 and S3 [2], [3]. The numbers of datasets and queries in the system are randomly drawn in the range of [50, 100] and [50, 150], respectively. The number of datasets required by the query is randomly drawn from interval [1, 3], respectively. The QoS



(a) The evaluation cost (b) The average delay (c) The average error bound of evaluation result

Fig. 3. The performance of different algorithms for a single query.



(a) The evaluation cost (b) The average delay (c) The average error bound of evaluation result

Fig. 4. The impact of the maximum number of datasets of a single query on the algorithmic performance.

requirement of each query is a value between 500 and 1,000 ms. Unless otherwise specified, we will adopt these default settings in our simulations. Each value in the figures is the mean of the results by applying each mentioned algorithm on 15 different topologies of the distributed cloud.

We evaluate the performance of algorithm `Appro_DPR` for a single approximate query against a greedy algorithm `Greedy`. Algorithm `Greedy` first tries to place the samples with the smallest error bounds and then adjusts the error bounds of samples until all datasets have placed samples in datacenters. To evaluate the performance of the proposed algorithm `Heu_DPR` for multiple approximate queries, another greedy algorithm `Greedy_multiQ` is employed as an evaluation baseline. Specifically, `Greedy_multiQ` algorithm first selects a set of candidate datacenters for each pair of a query and one of its requested datasets. If the delay requirement of the query can be met by putting a sample with the smallest error bound of the dataset at the selected datacenters, it then places the sample at a datacenter with the maximum available computing resource. If the datacenter cannot accommodate more samples, it then picks the next datacenter with the second largest amount of available computing resource in the set of candidate datacenters. If the available computing resource of the set of candidate datacenters cannot accommodate more samples, the algorithm then increases the error bounds of samples until all samples can be placed.

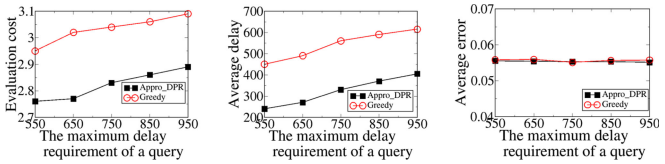
In addition, we evaluate the proposed algorithms on a real testbed. Motivated by the work [38] of Yan et al. that made better use of Amazon T2 instances, we lease a number of computing resources located at geo-distributed locations, and deploy a distributed computing environment, based on which we evaluate the proposed algorithms `Appro_DPR` and `Heu_DPR` against two baselines `Greedy` and `Greedy_multiQ`, respectively.

5.2 Performance Evaluation of Different Algorithms for a Single Query

We first evaluate the performance of algorithm `Appro_DPR` against that of algorithm `Greedy` in terms of the evaluation cost (the unit is US dollars), the average delay experienced by a query, and the average error bound of the evaluation result, by varying the number of datacenters from 20 to 200. From Fig. 3a it can be seen that the evaluation cost by algorithm `Appro_DPR` is much lower than that by algorithm `Greedy`. The reason is that algorithm `Appro_DPR` finds samples with the lowest error bound that can be dealt with by the resources of datacenters in the distributed cloud and then assigns queries to the datacenters. However, algorithm `Greedy` assigns queries to datacenters one by one and greedily increases the error bounds of samples until they are placed. In addition, the evaluation cost of each query by the two algorithms is increasing with the growth of the number of datacenters from 20 to 150. The rationale is that each query has a higher probability of requesting a dataset that is far from its home datacenter in the distributed cloud with more datacenters, thereby increasing the costs incurred by data transmission and sample updates. However, the evaluation cost decreases slightly when the number of datacenters grows from 150 to 200. The reason is that queries have more choices to select datacenters that are close to their home datacenters, which traded-off the mentioned cost increase due to the growth of datacenter numbers. Also, we can see from Fig. 3b, the average delay of queries by algorithm `Greedy` is much higher than that by algorithm `Appro_DPR`, because algorithm `Greedy` assigns queries one by one greedily and this may lead to the later-assigned queries being assigned to the datacenters far from their home datacenters (with higher access and update delays). Furthermore, as shown in Fig. 3c, the average error bound of all placed samples by algorithm `Appro_DPR` is almost the same as that of algorithm `Greedy`.

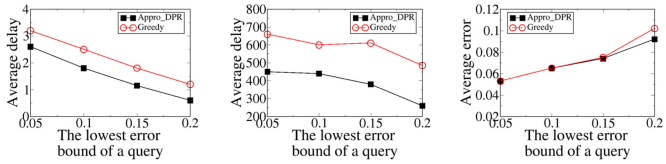
We then investigate the performance of algorithms `Appro_DPR` and `Greedy` by varying the maximum number of datasets of a query from 5 to 50 while fixing the number of datacenters at 20. From Figs. 4a and 4b, we can see that the curves of evaluation costs and the average delays of both algorithms are increasing with the growth of the maximum number of datasets, because more datasets usually incur higher process, storage, transmission, and update costs for a query, and increase the maximum delay of replicating their samples to assigned datacenters. As shown in Fig. 4c, the average error bound of all placed samples by algorithm `Appro_DPR` is almost the same as that of algorithm `Greedy`.

We third study the performance of algorithms `Appro_DPR` and `Greedy` by varying the maximum delay requirement of a query and fixing the number of datacenters at 20, as illustrated in Fig. 5. From Fig. 5a, we can see that algorithm `Appro_DPR` delivers a solution with a much lower evaluation cost than that of algorithm `Greedy`. However, both of their evaluation costs are increasing as the maximum delay requirement increases. This is because a higher maximum delay requirement of a query (1) means the query can wait longer time to gain an evaluation result with a lower error bound, increasing the process and storage costs; (2) allows the algorithms to put its samples to far away datacenters from their home datacenters, thereby increasing the update and transmission costs. Fig. 5b demonstrates that the average delay of a query is increasing with the growth of its maximum delay requirement. The reason is similar as that of Fig. 5a. In addition, we can see from Fig. 5c that the curves for



(a) The evaluation cost (b) The average delay (c) The average error bound of evaluation result

Fig. 5. The impact of the maximum delay requirement of a single query on the algorithmic performance.



(a) The evaluation cost (b) The average delay (c) The average error bound of evaluation result

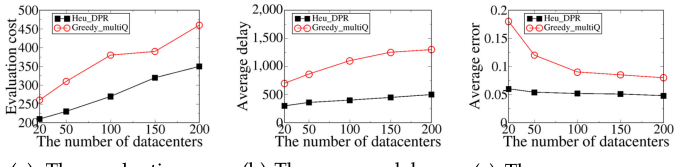
Fig. 6. The impact of the lowest error bound of a single query on the algorithmic performance.

the average error of a query evaluation result are slightly decreasing as the maximum delay requirement increases. The rationale behind is that a query usually has a stringent requirement on its average error bound of its results while it has a loose requirement on delay.

We finally evaluate algorithm `Appro_DPR` against algorithm `Greedy`, by increasing the lowest error bound of a sample from 0.05 to 0.15, while fixing the maximum error bound of a sample at 0.25. From Figs. 6a and 6b, we can see that the evaluation costs and the average delay of a query by both algorithms are decreasing when the lowest error bound of a sample increases from 0.05 to 0.15. The reason is that a higher error bound of a sample usually means a smaller size of the sample, thereby leading to a lower cost and shorter time of evaluating the query. Furthermore, since evaluating a query based on smaller volume of samples usually incurs a larger error, the average error of a query evaluation result is increasing with the growth of the lowest error bound of its data samples, as depicted in Fig. 6c.

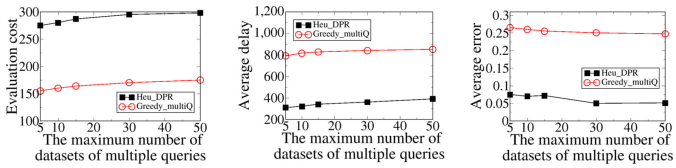
5.3 Performance Evaluation of Different Algorithms for Multiple Queries

We now evaluate the proposed algorithm `Heu_DPR` against algorithm `Greedy_multiQ`, in terms of the evaluation cost (US dollars), the average delay, and the average error bound of the evaluation result, by varying the number of datacenters from 20 to 200. It can be seen from Fig. 7a that the evaluation cost by algorithm `Heu_DPR` is substantially less than that by algorithm `Greedy_multiQ`. For example, the evaluation cost by `Heu_DPR` is only about 77 percent of that by algorithm `Greedy_multiQ` when there are 20 datacenters. The rationale behind is that algorithm `Greedy_multiQ` selects the datacenter with the highest available computing resource to place samples which may be too far to satisfy the delay requirement of queries, thus more slave samples are placed to satisfy the delay requirement of queries, increasing storage, update and transmission costs. It can also be seen that the evaluation costs of both algorithms increase with the growth of the number of datacenters. One



(a) The evaluation cost (b) The average delay (c) The average error bound of evaluation result

Fig. 7. The performance of different algorithms for multiple queries.

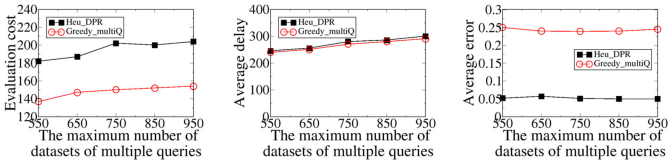


(a) The evaluation cost (b) The average delay (c) The average error bound of evaluation result

Fig. 8. The impact of the maximum number of datasets of queries on the algorithmic performance.

of the main reasons is that with the growth of the number of datacenters, more computing resource will be available to accommodate large-volume slave samples with lower error bounds, thereby increasing the costs of processing, storing, transmitting and updating larger slave samples. From Fig. 7b we can see that the average delay by algorithm `Heu_DPR` is much lower than that by algorithm `Greedy_multiQ`, and the average delay of a query by both algorithms increases with the growth of number of datacenters. This is because the samples of a query have a higher probability of being assigned to datacenters that are far from its home datacenter. In addition, we can see from Fig. 7c that the average error bound of all samples placed in the system by algorithm `Heu_DPR` is only 0.07, while the one by algorithm `Greedy_multiQ` is 0.19 when the number of datacenters is 20, which is much higher than that by algorithm `Heu_DPR`. The reason is that algorithm `Heu_DPR` adopts a fine-grained adjustment of error bounds when there exist queries that cannot be evaluated. On the other hand, algorithm `Greedy_multiQ` places more slave samples for each origin sample, thus, occupies more computing resource. This prevents the algorithm from placing slave samples with lower error bounds that typically have higher computing resource demands. Also, the average error bound of all evaluated queries by algorithm `Greedy_multiQ` decreases with the growth of the number of datacenters, since more resources in the datacenters can be used to process samples with larger sizes and lower error bounds.

We then investigate the impact of the maximum number of datasets of queries on the performance of algorithms `Heu_DPR` and `Greedy_multiQ`, by varying the maximum number of datasets from 5 to 50 and fixing the number of datacenters to 20. It can be seen from Fig. 8a that the evaluation cost of all queries by algorithm `Greedy_multiQ` is lower than that by algorithm `Heu_DPR`. For example, the evaluation cost by algorithm `Greedy_multiQ` is 58 percent of that by algorithm `Heu_DPR`. While we can see from Figs. 8b and 8c that the average delay experienced by a query and the average error bound of evaluation result of the query by algorithm `Greedy_multiQ` are much higher



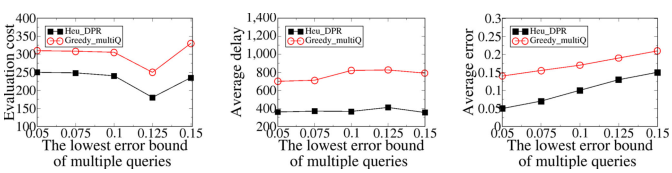
(a) The evaluation cost (b) The average delay (c) The average error bound of evaluation results

Fig. 9. The impact of the maximum delay requirement of multiple queries on the algorithmic performance.

than those by algorithm Heu_DPR. For example, the average delay and the average error bound by algorithm Greedy_multiQ are 2.3 and 5 times higher than those by algorithm Heu_DPR. We can also see that the evaluation costs of both algorithms do not vary too much when the maximum number of datasets grows. The reason is that both algorithms consider the assignment of multiple queries, and later assigned queries can use the samples that are already placed by earlier assigned queries. Although the maximum number of datasets of each query is increasing, most of their samples may already have been placed to evaluate early assigned queries, thereby saving the storage, transmission, and update costs. Similarly, the average delay and the average error bound of a query do not change too much with the growth of the maximum number of datasets, as shown in Figs. 8b and 8c, respectively.

We third study the impact of the maximum delay requirement of queries on the performance of algorithms Heu_DPR and Greedy_multiQ, by varying it from 550 to 950 ms and fixing the number of datacenters to 20. From Figs. 9a and 9c, it can be seen that although algorithm Heu_DPR obtained a higher evaluation cost than algorithm Greedy_multiQ, it delivers an evaluation result with much lower average error bound compared with Greedy_multiQ. For example, the evaluation cost by Heu_DPR is 1.28 times higher that that by Greedy_multiQ, but the average error bound of query evaluation result by Heu_DPR is 20 percent of that by Greedy_multiQ. The reason is that a lower average error usually means larger volumes of samples, which incurs higher costs in processing, storing, transmitting, and updating data. Also, the evaluation cost is increasing with the growth of the maximum delay requirement of a query as shown in Fig. 9a, the argue is similar as that for Fig. 5a. Furthermore, due to the fact that smaller volumes of samples are processed by algorithm Greedy_multiQ, it has a lower average delay as shown in Fig. 9b.

We finally evaluate the impact of the lowest error bound of samples on the performance of algorithms Heu_DPR and Greedy_multiQ through varying it from 0.05 to 0.15.



(a) The evaluation cost (b) The average delay (c) The average error bound of evaluation result

Fig. 10. The impact of the lowest error bound of multiple queries on the algorithmic performance.

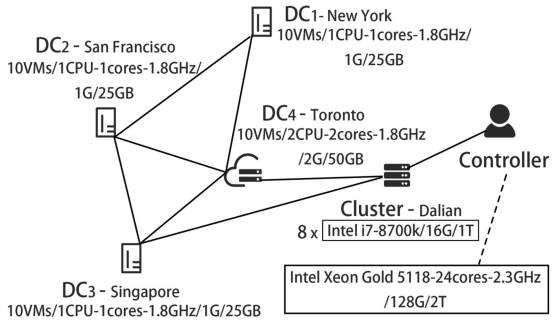


Fig. 11. The topology of the test-bed with leased VMs, a local server cluster, and a local SDN controller.

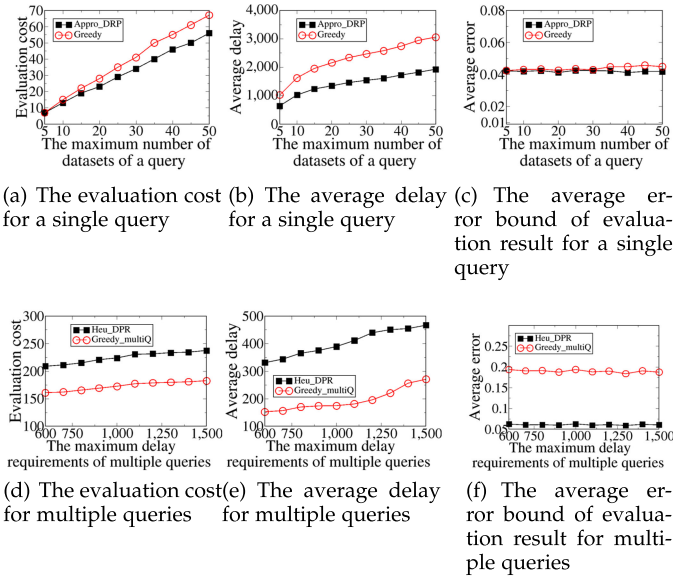
From Figs. 10a and 10c, a clear trade-off between the evaluation cost and the lowest error bound can be seen. Also, in general the evaluation costs by algorithms Heu_DPR and Greedy_multiQ decrease with the increase of the maximum error bound from 0.05 to 0.125. The rationale is that with the growth of the lowest error bound of samples, samples with smaller sizes need to be processed and transmitted. However, the evaluation cost increases suddenly when the lowest error bound increases from 0.125 to 0.15. This is because with the growth of the lowest error bound of a sample, queries will be less likely to use it, as the average error bound needs to be minimized as well. Therefore, the samples with higher error bounds have a high probability of not being used by later queries, thereby increasing the storage cost of storing the needed samples with lower error bounds.

5.4 Performance Evaluation in a Real Test-Bed

We now evaluate the performance of the proposed algorithms in a real test bed, the test bed is composed of virtual machines, a local server cluster, and a controller. The virtual machines are located at different locations and provided by a cloud service provider, while the controller is responsible for executing the proposed algorithms

Test-Bed Settings. As shown in Fig. 11, we lease a number of virtual machines (VMs) at locations San Francisco, New York, Toronto, and Singapore from cloud service provider DigitalOcean [12]. It must be mentioned that since we focus on the inter-datacenter scheduling of approximate queries, we use 10 VMs to represent a datacenter in the distributed cloud network G. Although the scale of each node in this test-bed can not be comparable to a large-scale datacenter, the implementation can be easily extended to a test-bed with large-scale computing nodes. In addition, we implemented a Container-based local cluster that consists of 8 servers, each server has an i7 quad-core, 8 Gigabyte memory. There is also an software defined network (SDN) controller implemented in a server with 24-core Intel Xeon Glod processor and 128 GB memory. The proposed algorithms are implemented as Java programs in the controller.

Datasets, Samples, and Approximate Queries. The data used in the experiment is mobile application usage information from 3,000 anonymous mobile users for a period of three months. We divide the data into a number of datasets according to the data creation time, and randomly distribute the datasets into the datacenters of the test-bed. Each dataset is sampled using stratified sampling, specifically, each dataset is divided into several groups, called strata, then a



(a) The evaluation cost for a single query (b) The average delay for a single query (c) The average error bound of evaluation result for a single query

(d) The evaluation cost for multiple queries (e) The average delay for multiple queries (f) The average error bound of evaluation result for multiple queries

Fig. 12. The performance of algorithms `Appro_DPR` and `Greedy` for a single query, the performance of algorithms `Heu_DPR` and `Greedy_multiQ` for multiple queries in a real test bed.

probability sample is drawn from each group. Approximate queries are issued to find some evaluation results: such as the most popular applications, at what time the found applications would be used in future, and the usage pattern of some mobile applications, etc.

Results. We evaluate the performance of the proposed algorithms `Appro_DPR` and `Heu_DPR` against benchmarks `Greedy` and `Greedy_multiQ`, respectively. Due to page limits, we here put only two sets of figures as illustrated in Fig. 12. From Figs. 12a and 12b we can see that, algorithm `Appro_DPR` outperforms algorithm `Greedy` by delivering much lower evaluation cost and average delay for an approximate query. The average error bound of evaluation results by algorithm `Appro_DRP` is slightly lower than that of algorithm `Greedy` as depicted in Fig. 12c; however, the gap increases as the number of datasets required by a query grows. From Figs. 12d and 12e, we can see that although the evaluation cost and average delay by `Heu_DPR` are higher than those by `Greedy_multiQ`, the average error bound of evaluation results is much lower than that by `Greedy_multiQ` as shown in Fig. 12f.

6 Related Work

Several studies of data placement and query evaluation have been conducted in the past [1], [7], [8], [15], [16], [17], [19], [23], [25], [27], [28], [30], [31], [32], [33], [34], [35], [37]. Most of these studies either did not take into consideration data replications of generated big data [15], [16], [23], [31] or did not incorporate the QoS requirements of users into account [7], [15], [17], [18], [19], [30], [31]. Few of them only considered the transmission cost while neglecting other costs [17], [18]. For example, Baev et al. [7] considered a problem of placing replicated data in arbitrary networks to minimize the total storage and access cost. Golab et al. [15] studied a data placement problem to determine where to store the data and where to evaluate data-intensive tasks with a goal to minimize the data transmission cost. Kayyoor et al. [17] addressed a problem of minimizing average query span, which is the number of servers involved in answering a query. They ignored other costs and QoS requirements of users [7], [15], [17], and did not consider data replications [15]. Agarwal et al. [1] proposed a data placement mechanism Volley for geo-distributed cloud services to minimize the user-perceived latency. Xia et al. [31] considered a big data management problem in distributed cloud environments to maximize the system throughput while minimizing the operational cost of service providers. No data replications and QoS requirements of users have been addressed in [1], [31]. Pu et al. [23] presented a system for low latency geo-distributed analytics, which used an heuristic to redistribute datasets among the datacenters prior to queries' arrivals, and placed the queries to reduce network bottlenecks during the query's execution. Heintz et al. [16] studied the tradeoff between a query response delay and the errors of the query results in streaming analytics in an architecture consisting of a single center and multiple edge servers. Xia et al. [28] considered an online query evaluation problem for big data analytics in distributed clouds, with an objective to maximize the query acceptance ratio while minimizing the transmission cost. They did not consider approximate query evaluation that strives for a nontrivial trade-off between the query and the error bounds of query results.

Some studies considered query evaluation or data sample problems [8], [16], [18], [19], [23], [25], [27]. For example, Li et al. [19] proposed a similarity aware data analytics system to minimize query completion time. No data samples and approximate queries are studied. Shkapsky et al. [25] implemented a system which is a recursive query language implementation on Apache Spark for big data analytics. Convolbo et al. [8] studied a problem where all jobs are submitted in a centralized global scheduler to be dispatched among some geo-distributed datacenters for processing. They assumed that the initial data locations and data replication strategy are given, and data analytic jobs can be divided into multiple independent sub-jobs. They aimed to minimize the average completion time needs to make scheduling decisions. Xiao et al. [27] proposed a framework that considered a problem of data movement, resource provisioning, and Reduce selection, their objective is to minimize the cost incurred by optimizing the amount of data allocated to each datacenter, the number of resources needed, and the datacenter for Reduce operation.

There are also some studies considered big data analysis and query evaluation in systematic perspectives [21], [26]. For example, Mehta et al. [21] presented evaluations of large-scale image analysis for which two real-world scientific image data processing use cases were adopted. Five representative systems, SciDB, Myria, Spark, Dask, and TensorFlow are evaluated, the shortcomings that complicate implementation or hurt performance are found, which lead to new research opportunities in making efficient and easy large-scale image analysis. This work contributed the first comprehensive and quantitative evaluation of large-scale image analysis systems. Upadhyaya et al. [26] developed an optimizer that can automatically selects strategy for each operator to minimize the total time to complete the query given an expected number of failures. They also modeled the processing and recovery times for some representative

operators (e.g., relational operators, maps or reduces). Authors in [29] considered a problem of data replication and placement for query evaluation of big data analytics, a heuristic algorithm for multiple queries is designed. However, the work did not consider that sometimes queries arrive at the system one-by one and there is only one query to be evaluated each time, and the work only evaluated the proposed algorithms by simulations.

Unlike the aforementioned studies, in this paper we studied the QoS-aware data replication and placement problem for query evaluation of big data analytics in distributed cloud environments, where big data are located at different locations and users have their QoS requirements in terms of query delays, with the objective to minimize the evaluation cost of all queries while meeting the QoSs of the users of these queries. We aim to evaluate the queries as fast as possible while keeping the query result as accurate as possible, we focus on the sample choice and placement on different datacenters in a distributed cloud. Cases of both a single query and multiple queries are considered, and the algorithms are evaluated on a real testbed based on real datasets.

7 Conclusions

In this paper, we studied approximate query evaluation of big data analytics in a distributed cloud, we formulated two novel query optimization problems: QoS-aware data replication and placement problems for a single approximate query and multiple approximate queries of big data analytics, respectively. Our aim is to minimize the query evaluation cost, while meeting the query delay requirements and computing resource capacity constraints of datacenters, through efficient and effective data replication and placement. To this end, we first devised an approximation algorithm for evaluating a single approximate query. We then proposed a fast yet scalable heuristic for multiple approximate query evaluation by reducing the problem to an unsplittable multicommodity flow problem. We finally evaluated the performance of the proposed algorithms through both experimental simulations and implementations in a test-bed. Experimental results demonstrate that the proposed algorithms are promising.

Acknowledgments

We would like to thank the three anonymous referees and the associate editor for their expertise comments and constructive suggestions, which have helped us improve the quality and presentation of the paper greatly. The work of Qiufen Xia and Zichuan Xu is partially supported by the National Natural Science Foundation of China (Grant No. 61802047, 61802048, 61772113, 61872053), the fundamental research funds for the central universities in China (Grant No. DUT19RC(4)035, DUT19RC(5)001, DUT19GJ204), and the "Xinghai Scholar" Program at Dalian University of Technology, China.

References

[1] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2010, pp. 2–2.

[2] [Online]. Available: https://aws.amazon.com/ec2/. Accessed on: Jun. 2019.

[3] [Online]. Available: https://aws.amazon.com/s3/, Accessed on: Jun. 2019.

[4] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "BlinkDB: Queries with bounded errors and bounded response times on very large data," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2013, pp. 29–42.

[5] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of datacenters for cloud computing," *J. Future Generation Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[6] P. A. Bernstein and S. Das, "Rethinking eventual consistency," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 923–928.

[7] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM J. Comput.*, vol. 38, no. 4, pp. 1411–1429, 2008.

[8] M. W. Convolbo, J. Chou, and S. Lu, "DRASH: A data replication-aware scheduler in geo-distributed data centers," in *Proc. Int. Conf. Cloud Comput. Technol. Sci.*, 2016, pp. 302–309.

[9] S. Chaudhuri, G. Das, and V. Narasayya, "Optimized stratified sampling for approximate query processing," *Trans. Database Syst.*, vol. 32, no. 2, pp. 1–50, 2007.

[10] Cisco Global Cloud Index: Forecast and Methodology, White paper, pp. 1–46.

[11] GT-ITM: Georgia tech internetwork topology models. [Online]. Available: www.cc.gatech.edu/projects/gtitm

[12] Digital Ocean. [Online]. Available: https://www.digitalocean.com. Accessed on: Nov. 2018.

[13] W. Fan, F. Geerts, and F. Neven, "Making queries tractable on big data with preprocessing: Through the eyes of complexity theory," *Proc. VLDB Endowment*, vol. 6, no. 9, pp. 685–696, 2013.

[14] B. Gendron, T. G. Crainic, and A. Frangioni, "Multicommodity capacitated network design," *Telecommunications Network Planning*. Berlin, Germany: Springer, 1999, pp. 1–19.

[15] L. Golab, M. Hadjieleftheriou, H. Karloff, and B. Saha, "Distributed data placement to minimize communication costs via graph partitioning," in *Proc. Int. Conf. Sci. Statist. Database Manage.*, 2014, Art. no. 20.

[16] B. Heintz, A. Chandra, and R. K. Sitaraman, "Trading timeliness and accuracy in geo-distributed streaming analytics," in *Proc. ACM Symp. Cloud Comput.*, 2016, pp. 361–373.

[17] A. K. Kayyoor, A. Deshpande, and S. Khuller, "Data placement and replica selection for improving co-location in distributed environments," *Comput. Res. Repository*, arXiv: 1302.4168, pp. 1–12, 2012.

[18] P. Li, S. Guo, T. Miyazaki, X. Liao, H. Jin, A. Y. Zomaya, and K. Wang, "Traffic-aware geo-distributed big data analytics with predictable job completion time," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1785–1796, Jun. 2017.

[19] H. Li, H. Xu, and S. Nutanong, "Bohr: Similarity aware geo-distributed data analytics," in *Proc. 9th USENIX Conf. Hot Topics Cloud Comput.*, 2017, pp. 2–2.

[20] R. Lu, H. Zhu, X. Liu, J. K. Liu, and J. Shao, "Toward efficient and privacy-preserving computing in big data era," *IEEE Netw.*, vol. 28, no. 4, pp. 46–50, Jul./Aug. 2014.

[21] P. Mehta, S. Dorkenwald, D. Zhao, T. Kaftan, A. Cheung, M. Balazinska, A. Rokem, A. Connolly, J. Vanderplas, and Y. AlSayyad, "Comparative evaluation of big-data systems on scientific image analytics workloads," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1226–1237, 2017.

[22] M. Noordzij, G. Tripepi, F. W. Dekker, C. Zoccali, M. W. Tanck, and K. J. Jager, "Sample size calculations: Basic principles and common pitfalls," *Nephrology Dialysis Transplantation*, vol. 25, no. 5, pp. 1388–1393, 2010.

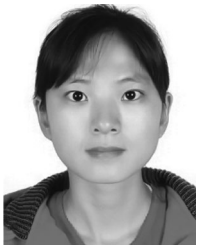
[23] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proc. ACM Conf. SIGCOMM*, 2015, pp. 1–14.

[24] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves, "Sailfish: A framework for large scale data processing," in *Proc. ACM Symp. Cloud Comput.*, 2012, Art. no. 4.

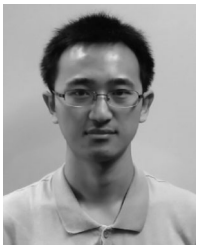
[25] A. Shkapsky, M. Yang, M. Interlandi, H. Chiu, T. Condie, and C. Zaniolo, "Big data analytics with datalog queries on Spark," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 1135–1149.

[26] P. Upadhyaya, Y. Kwon, and M. Balazinska, "A latency and fault-tolerance optimizer for online parallel query plans," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 241–252.

[27] W. Xiao, W. Bao, X. Zhu, and L. Liu, "Cost-aware big data processing across geo-distributed datacenters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3114–3127, Nov. 2017.
[28] Q. Xia, W. Liang, and Z. Xu, "Data locality-aware big data query evaluation in distributed clouds," *Comput. J.*, vol. 60, no. 6, pp. 791–809, 2017.
[29] Q. Xia, W. Liang, and Z. Xu, "QoS-Aware data replications and placements for query evaluation of big data analytics," in *Proc. IEEE Int. Conf. Commun.*, 2017, pp. 1–7.
[30] Q. Xia, W. Liang, and Z. Xu, "The operational cost minimization in distributed clouds via community-aware user data placements of social networks," *Comput. Netw.*, vol. 112, pp. 263–278, 2017.
[31] Q. Xia, Z. Xu, W. Liang, and A. Zomaya, "Collaboration- and fairness-aware big data management in distributed clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 7, pp. 1941–1953, Jul. 2016.
[32] Z. Xu and W. Liang, "Minimizing the operational cost of data centers via geographical electricity price diversity," in *Proc. 6th IEEE Int. Conf. Cloud Comput.*, 2013, pp. 99–106.
[33] Z. Xu and W. Liang, "Operational cost minimization for distributed data centers through exploring electricity price diversity," *Comput. Netw.*, vol. 83, pp. 59–75, 2015.
[34] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao, "Task offloading with network function services in a mobile edge-cloud network," *IEEE Trans. Mobile Comput.*, doi: 10.1109/TMC.2018.2877623, vol. PP, no. 99, pp. 1–1, 2019.
[35] Z. Xu, W. Liang, and Q. Xia, "Electricity cost minimization in distributed clouds by exploring heterogeneities of cloud resources and user demands," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2015, pp. 388–395.
[36] Y. Yan, L. J. Chen, and Z. Zhang, "Error-bounded sampling for analytics on big sparse data," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1508–1519, 2014.
[37] S. Yu, M. Liu, W. Dou, X. Liu, and S. Zhou, "Networking for big data: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 531–549, Jan.–Mar. 2017.
[38] F. Yan, L. Ren, D. J. Dubois, G. Casale, J. Wen, and E. Smirni, "How to supercharge the Amazon T2: Observations and suggestions," in *Proc. IEEE 10th Int. Conf. Cloud Comput.*, 2017, pp. 278–285.



Qiufen Xia received the BSc and ME degrees in computer science from the Dalian University of Technology (DUT) in China, in 2009 and 2012, respectively, and the PhD degree in computer science from the Australian National University, in 2017. She is currently a lecturer with the International School of Information Science and Engineering, DUT. Her research interests include mobile cloud computing, query evaluation, big data analytics, big data management in distributed clouds, and cloud computing. She is a member of the IEEE.



Zichuan Xu (M'17) received the BSc and ME degrees in computer science from the Dalian University of Technology (DUT) in China, in 2008 and 2011, respectively, and the PhD degree in computer science from the Australian National University, in 2016. He is currently an associate professor with the School of Software, DUT. Before joining in DUT, he was a research associate with the Department of Electronic and Electrical Engineering, University College London, United Kingdom. His research interests include cloud computing, software-defined networking, wireless sensor networks, routing protocol design for wireless networks, algorithmic game theory, and optimization problems. He is a member of the IEEE.

Weifa Liang (M'99–SM'01) received the BSc degree in computer science from Wuhan University, China, in 1984, the ME degree in computer science from the University of Science and Technology of China, in 1989, and the PhD degree in computer science from the Australian National University, in 1998. He is currently a full professor with the Research School of Computer Science, Australian National University. His research interests include design and analysis of energy efficient routing protocols for wireless ad hoc and sensor networks, cloud computing, software-defined networking, design and analysis of parallel and distributed algorithms, approximation algorithms, combinatorial optimization, and graph theory. He is a senior member of the IEEE.

Shui Yu received the BEng (electronic engineering) and MEng (computer science) degrees from the University of Electronic Science and Technology of China, P. R. China, in 1993 and 1999, respectively, and the PhD (computer science) degree from Deakin University, in 2004. He is currently a full professor with the School of Software, University of Technology Sydney (UTS), Australia. Before joining UTS, he was a senior lecturer with the School of Information Technology, Deakin University, Melbourne. His research interests include security and privacy, networking, big data, and mathematical modeling. He is a member of the AAAS, ACM, and a senior member of the IEEE.

Song Guo received the PhD degree in computer science from the University of Ottawa. He is currently a full professor with the Hong Kong Polytechnic University. Prior to joining PolyU, he was a professor of the University of Aizu, Japan. His research interests are mainly in the areas of cloud computing, big data, wireless network, and cyber-physical system. He has authored/edited seven books and more than 300 papers in refereed journals and conferences in these areas. He serves/ served in editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Emerging Topics in Computing*, the *IEEE Communications Magazine*, the *Wireless Networks*, and many other major journals. He has been the general/program chair or in organizing committees of numerous international conferences. He is a senior member of the IEEE and ACM, and an IEEE Communications Society distinguished lecturer.

Albert Y. Zomaya (M'90–SM'97–F'04) is currently the chair professor of High Performance Computing and Networking with the School of Computer Science, University of Sydney, Sydney, NSW, Australia, where he is also the director of the Centre for Distributed and High Performance Computing, which was established in 2009. He has authored more than 500 scientific papers and articles, and has authored, co-authored, or edited more than 20 books. He serves as an associate editor of 22 leading journals, such as, the *ACM Computing Surveys* and the *Journal of Parallel and Distributed Computing*. He was a recipient of the IEEE Technical Committee on Parallel Processing Outstanding Service Award (2011), the IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing (2011), and the IEEE Computer Society Technical Achievement Award (2014). He is a chartered engineer, and a fellow of the American Association for the Advancement of Science and the Institution of Engineering and Technology (United Kingdom). His research interests include the areas of algorithms, parallel and distributed computing, and complex systems. He is a fellow of the IEEE.