This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing

Trading Cost and Throughput in Geo-distributed Analytics with A Two Time Scale Approach

Xinping Xu, Wenxin Li, Renhai Xu, Heng Qi, Keqiu Li, *Senior Member, IEEE,* and Xiaobo Zhou

Abstract—In the era of global-scale services, analytical queries are performed on datasets that span multiple data centers (DCs). Such geo-distributed queries generate a large amount of inter-DC data transfers at run time. Due to the expensive inter-DC bandwidth, various methods have been proposed to reduce the traffic cost in geo-distributed data analytics. However, current methods do not attempt to address the throughput issue in geo-distributed analytics. In this paper, we target at characterizing and optimizing a cost-throughput tradeoff problem in geo-distributed data analytics. Our objectives are two-fold: (1) we minimize the inter-DC traffic cost when serving geo-distributed analytics with uncertain query demand, and (2) we maximize the system throughput, in terms of the number of query requests that can be successfully served with guaranteed queuing delay. Specifically, we formulate a stochastic optimization problem that seamlessly combines these two objectives. To solve this problem, we take advantage of Lyapunov optimization techniques to design and analyze a two-timescale online control framework. Without prior knowledge of future query requests, this framework makes online decisions on input data placement and admission control of query requests. Rigorous theoretical analyses show that our framework can achieve a near-optimal solution and maintain system stability and robustness as well. Extensive trace-driven simulation results further demonstrate that our framework is capable of reducing inter-DC traffic cost, improving system throughput, and guaranteeing a maximum delay for each query request.

Index Terms—Geo-distributed data analytics, Data placement, Admission control, Lyapunov optimization, Two-timescale Approach.

1 Introduction

Globe-scale organizations (e.g., Google [1] and Microsoft [2]) construct multiple data centers (DCs) across the world to deliver their services. These services continuously produce large volumes of data when logging user activity or monitoring server status [3–5]. These data are born and stored in multiple DCs, and introduce an interesting research topic of geo-distributed data analytics (GDA) [6, 7]. Such GDA has a huge impact on the business process and is of great importance to service providers. For example, GDA enables service providers to make advertisement decisions by querying user logs and detect attacks/faults by querying system logs.

Due to the geo-distributed feature of these datasets, GDA incurs substantial inter-DC traffic cost. GDA queries generate a large amount of inter-DC transfers, no matter they are performed in a centralized or distributed way. The centralized way aggregates all datasets to a single DC before executing queries, while distributed way leaves data in-place and executes queries directly on these geo-distributed datasets. Clearly, both of the two ways generate a large volume of inter-DC transfers, caused by either the input data transfers or intermediate data transfers. In fact, the amount of daily inter-DC transfers can be up to tens or hundreds of TBs in typical global-scale organizations (e.g., Facebook [8], Twitter [9], Yahoo [10]). While the inter-DC traffic is increasing, inter-DC communication is expensive. For example, in Amazon EC2, the cost of two machines communicating at the rate of the average WAN bandwidth between two DCs is up to 38× of the cost of renting these two machines [11].

From the economics perspective, it is the need to reduce or even minimize the inter-DC traffic cost incurred by GDA queries. Nevertheless, this imposes challenges in maximizing the throughput (the number of successfully served query requests), which is an important performance metric in the modern GDA system. Typically, a GDA system not only needs to handle millions of query requests per minute today. It also must be able to handle the ever-increasing number of query requests in the future [12]. Purely minimizing inter-DC traffic cost could result in low throughput, and leads to high queuing delay for each query request due to the scarce inter-DC bandwidth [7]. Even worse, purely optimizing system throughput can arbitrarily increase traffic cost. This is because of the fundamental difference between the two metrics: cost saving is usually obtained by reducing the number of successfully served query requests, whereas throughput improvements incur an increasing amount of inter-DC traffic.

Hence, we believe that it is crucial to characterize and optimize a *cost-throughput* tradeoff for a GDA system. *In other words, how to minimize the involved traffic cost while maximizing the number of query requests that can be served with guaranteed delay?* Intuitively, it may be a step towards the right direction to design an offline algorithm to obtain the optimal solution. However, such offline optimization inevitably relies on prior knowledge of future query demand. The query demand mainly refers to the number of query requests in each time, and it is typically uncertain over time.

- *X. Xu, W. Li, H. Qi and K. Li are with the School of Computer Science and Technology, Dalian University of Technology, No 2, Linggong Road, Dalian 116023, China. E-mail: pixxinger@mail.dlut.edu.cn, toliwenxin@gmail.com, {hengqi, keqiu}@dlut.edu.cn.*
- *R. Xu and X. Zhou are with the Tianjin Key Laboratory of Advanced Networking, College of Intelligence and Computing, Tianjin University, Tianjin 300350, P.R. China. E-mail: {xurenhai, xiaobo.zhou}@tju.edu.cn.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing

Moreover, the arrivals of query requests may not follow any stationary distributions, yet are not known a prior.

To the best of our knowledge, no existing work is in place to solve such *cost-throughput* problem for GDA queries with uncertain demand. State-of-the-art methods can be divided into two categories: *1)* The first one is to reduce the inter-DC traffic cost by leveraging efficient data replication, and task aggregation strategies for GDA queries [7, 13, 14]; *2)* The second one is to shorten the completion time of GDA query jobs via efficient data/task placement and flow scheduling strategies [6, 15–17]. However, all of them ignore the throughput maximization in the GDA system. One may wonder at this point that reducing the completion of GDA queries indirectly leads to high throughput. Nonetheless, existing methods suffer from assumptions that are often unrealistic—such as analytical queries are repeated or known in advance.

In this paper, we take the first step to study the *cost-throughput* problem for large-scale GDA systems, in the presence of uncertain query demand. Our primary focus is to *minimize the inter-DC traffic cost* and *maximize the number of query requests that can be successfully served with guaranteed delay*. To this end, we blend the advantages of both input data placement and query request admission techniques. Especially, we formulate a stochastic optimization problem, which takes into account three constraints: ensuring the completion of input data placement before queries arrive, guaranteeing a maximum delay for each query request, and bounding the involved inter-DC traffic within the capacity. To solve this problem, we design a practical and provably-efficient two-timescale online control framework based on the Lyapunov optimization technique. Without prior knowledge of future query requests, this framework makes online decisions on the input data placement in time slots of longer periods of time, and also determines the number of query requests that can be served with guaranteed delay in smaller time scales. The key is that such a two-timescale framework can adaptively re-distribute input data among different DCs before starting the query requests. This will reduce the traffic cost in the long-term when continuously receiving a large number of query requests, and at the same time, accommodate more query requests in the inter-DC network. We use rigorous mathematical analysis to demonstrate that our framework can achieve a near-optimal solution and maintain system stability and robustness. We further conduct extensive simulations based on the 7-day worth of traces from Google Cluster Usage, to show the effectiveness of our framework in reducing inter-DC traffic cost, improving system throughput, and guaranteeing a maximum delay for each query request.

The main contributions of this paper include:

- We take the first step to study a *cost-throughput* tradeoff problem in geo-distributed data analytics without any prior knowledge of future query requests' arrival patterns. Specifically, we formulate a stochastic optimization problem, which takes into account the inter-DC traffic cost, system throughput, and the maximum queuing delay.
- We present a two-timescale online control framework to solve the stochastic problem. The key to optimizing the *cost-throughput* tradeoff in this framework is that it adaptively adjusts the input data distribution among different DCs at the beginning of a longer time scales, and determines how many query requests to be simultaneously served at the beginning of each smaller time scales.
- We conduct rigorous theoretical analysis as well as large-scale trace-driven simulations to evaluate the performance of our proposed two-timescale framework. Both theoretical and simulation results have shown that our framework can reduce inter-DC traffic cost, improving system throughput, and guaranteeing a maximum delay.

2 Background and Problem Statement

In this section, we first explain the geo-distributed data analytics and then emphasize the problem this paper studied.

2.1 Background

2.1.1 Geo-distributed Data Analytics

A geo-distributed analytics system logically spans multiple DCs. All DCs are connected to a non-blocking network, via dedicated uplinks and downlinks. The bottlenecks are only between DCs and the non-blocking core, which is reasonable because of recent studies and measurements [6, 18]. In addition, the bandwidth capacity varies significantly across different uplinks and downlinks. Data can be generated on any DCs, and as such, a dataset could be distributed across many DCs. Here, each dataset can only be used for one type of query requests. For example, querying for making advertisement decisions requires user logs, while querying for detecting DoS attacks requires network logs.

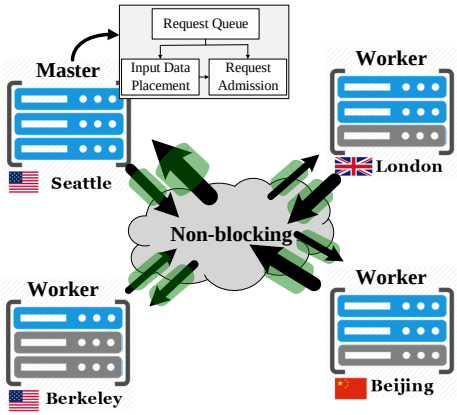


Fig. 1. An illustrative example of a GDA system.

Fig. 1 illustrates an example of a GDA system, where newly arrived query requests are stored in the corresponding queues. The *Master* DC keeps track of input data locations across multiple *Worker* DCs, makes decisions on input data placement, and determines how many query requests can be simultaneously executed each time. Once a query request has been decided to be executed, the *Master* DC converts this request's script into a job (e.g., a MapReduce [19] or Spark [20] job), which consists of many parallel tasks. These tasks are then pushed to multiple *Worker* DCs, for executing the job to obtain the final query results.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing

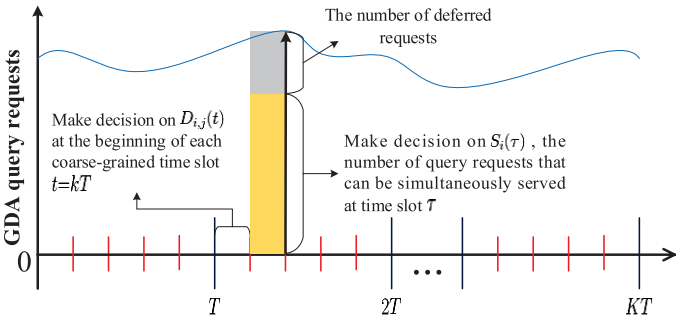


Fig. 2. An example of two-timescale GDA. Each coarse-grained time slot is divided into 5 fine-grained time slots (i.e., $T = 5$). When $T = 1$, our model is reduced to single time scale.

2.1.2 *Query Requests*

Each query request is viewed as a job such as MapReduce or Spark job. Input tasks of these query requests (e.g., map tasks) are executed on their corresponding input data, and write their outputs to their respective localities. These outputs are then fetched by a number of reduce tasks, which leads to a large amount of *intermediate data*. In the geo-distributed setting, the main aspect of cost is massive amounts of *intermediate data* that need to be transferred across multiple DCs. In fact, it has been reviewed that despite the local aggregation of map outputs, the ratio of intermediate to input data sizes in Facebook's analytics cluster is still high, 0.55 on average, and 24% of jobs have this ratio ≥ 1 [21, 22].

On the other hand, query requests can continuously arrive. The number of query requests in a GDA system is as high as millions per minute and is ever-increasing in the future. Combining a large amount of *intermediate data* generated by each query request, the cost will be extremely high if the input data are not properly placed because the distribution of input data significantly carries over to the distribution of *intermediate data*.

2.2 Problem Statement

The problem we studied is to design an algorithm to make decisions on both the input data placement and the admission control of query requests. The primary objective is to minimize the incurred inter-DC traffic cost and maximize the number of query requests that can be served with guaranteed queuing delay. The *input data placement*, determining how much amount of input data should be placed on each DC for each query type, is executed in time slots of longer periods of time. The query request admission, deciding the number of query requests that can be simultaneously executed, is performed in smaller time scales.

In this problem, information about future query requests is not known. Therefore, an online algorithm is desired. In the following, we start with mathematically formulating the problem and then leverage Lyapunov optimization techniques to design a two-timescale online control framework.

3 Cost-Performance Tradeoff Model

We consider a GDA system to logically span a set of DCs, $\mathcal{M}$=$\{1, 2, \ldots, M\}$. For each DC j, let U_j^u and U_j^d denote the uplink and downlink bandwidth capacities, respectively.

TABLE 1
Notations and Definitions

Symbol	Definition
M	The number of DCs, $j = 1, 2, \ldots, M$
N	The number of GDA query types, $i = 1, 2, \ldots, N$
K	The number of coarse-grained time slots
T	The number of fine-grained time slots in each coarse-grained time slot
U_j^u	The uplink bandwidth capacity for DC j
U_j^d	The downlink bandwidth capacity for DC j
$\hbar$	The time-length of each fine-grained time slot
D_i	The total amount of input data for i-th query type
$D_{i,j}(t)$	The amount of input data placed on DC j at time t, for i-th type of GDA query.
α_i	The ratio of intermediate data to input data, for i-th type of GDA query.
$A_i(\tau)$	The number of arrived requests of i-th type of GDA query at time τ.
$S_i(\tau)$	The number of served requests of i-th type of GDA query at time τ.
$Q_i(\tau)$	The queue backlog for i-th type of GDA query at time τ
$f_{i,j}^1(t)$	The amount of input data to be moved out of DC j at time t
$f_{i,j}^{1'}(t)$	The amount of input data to be moved in DC j at time t
$f_{i,j}^2(\tau)$	The amount of intermediate data to be uploaded by DC j at time τ
$f_{i,j}^{2'}(\tau)$	The amount of intermediate data to be downloaded by DC j at time τ

There are N query types, $\mathcal{N}$=$\{1, 2, \ldots, N\}$. Specifically, let D_i denote the total amount of input data for i-th query type. Inspired by the modeling work in DC networks [23, 24], we consider the system to operate in a discrete-time mode, where the time can be divided into K coarse-grained time slots. Each coarse-grained time slot is further divided into T fine-grained time slots. Each fine-grained time slot has the same duration $\hbar$, typically 5 or 15 minutes. Fig. 2 shows an example of such a two-timescale GDA system. Table 1 list important notations used throughout this paper.

3.1 Control decisions

3.1.1 *Input data placement*

At the beginning of each coarse-grained time slot t=kT(k=$1, 2, \ldots, K$), the first control decision is to determine how much amount of input data should be placed on each DC for each query type. Specifically, let $D_{i,j}(t)$ denote such decision variable with respect to j-th DC and i-th query type.

3.1.2 *Query request admission*

In each fine-grained time slot τ, a number $A_i(\tau)$ of requests of i-th query type arrive, and are stored in a queue $Q_i(\tau)$. Accordingly, another control decision is to determine how many requests, $S_i(\tau)$, can be simultaneously executed for each query type, in time slot τ. The rest of the requests are then deferred to later times with more available link bandwidth or lower traffic cost. So, we have the following *queuing dynamics* over time for each query type.

$$Q_i(\tau + 1) = \max\{Q_i(\tau) - S_i(\tau), 0\} + A_i(\tau), \forall \tau. \quad (1)$$

These queues takes $A_i(\tau)$ as input and $S_i(\tau)$ as output. $Q_i(\tau)$ is called as the backlog at time τ, as it represents the number of unserved query requests at time τ.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing

3.2 Constraints

The query request arrival rate $A_i(\tau)$ are random variables *without any probabilistic assumptions*. The following constraints should be satisfied.

3.2.1 Ensuring the completion of input data placement

The input data placement must be completed before executing the query requests. Specifically, let $f^1_{i,j}(t)$ (or $f^{1'}_{i,j}(t)$) denote *the amount of input data* to be moved out of (or to be moved in) DC j for i-th query type in time slot t.

$$f^1_{i,j}(t) = \max(D_{i,j}(t-T) - D_{i,j}(t), 0), \tag{2}$$

$$f^{1'}_{i,j}(t) = \max(D_{i,j}(t) - D_{i,j}(t-T), 0). \tag{3}$$

We enforce the input data movement to be completed within one fine-grained time slot. Thus, we have

$$\sum_{i=1}^{N} \frac{f^1_{i,j}(t)}{U^u_j} \le \hbar, \forall j, \forall t, \tag{4}$$

$$\sum_{i=1}^{N} \frac{f^{1'}_{i,j}(t)}{U^d_j} \le \hbar, \forall j, \forall t, \tag{5}$$

$$\sum_{j=1}^{M} D_{i,j}(t) = D_i, \forall i, \forall t. \tag{6}$$

Eq. (4) constrains the outward data movement, while Eq. (5) enforces the inward data movement. No matter how to move the input data, the total amount of input data for each query type should be fixed, as shown in Eq. (6). During these data movements, no query requests can be served. This is exactly described in the following two constraints.

$$S_i(\tau) = 0, \forall i, \forall \tau{=}t, \forall t, \tag{7}$$

$$0 \le S_i(\tau) \le Q_i(\tau), \forall i, \forall \tau{=}t{+}1, \ldots, t{+}T{-}1, \forall t. \tag{8}$$

It should be noted that the input data movement brings a negative impact on system throughput. However, it is performed every T fine-grained time slots and can be finished within merely one time slot. Hence, it will not impact the system throughput too much, as query requests may be admitted at each time slot (except the one for data movement). By setting a larger value of the parameter T, such impact can further be reduced.

3.2.2 Guaranteeing queuing delay of query requests

The queuing delay is closely related to the queue backlog, and hence we bound the length of the queue backlog. This, in turn, determines the delay performance for each query request. Let $\overline{Q} \triangleq \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}|Q_i(\tau)|$ denote the time-averaged expected backlog. To guarantee a maximum delay l_{max}, we consider the following two constraints

$$\overline{Q} = \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}|Q_i(\tau)| < \infty, \tag{9}$$

$$Q_i(\tau) < Q_{max}, \forall i, \forall \tau, \tag{10}$$

where Q_{max} is the maximum backlog. The maximum delay l_{max} is a proportional function of Q_{max}, which we will show in Section 4.1. So, when the queue backlog is bounded, the maximal delay can also be guaranteed [23].

3.2.3 Link capacity constraint

Both the uplink and downlink bandwidth capacities should be satisfied when transferring the *intermediate data* generated by query requests. Specifically, define $f^2_{i,j}(\tau)$ as the amount of intermediate data to be uploaded by DC j for i-th type of GDA query in each fine-grained time τ. Then, we have

$$f^2_{i,j}(\tau) = D_{i,j}(t)\alpha_i(1 - \frac{D_{i,j}(t)}{D_i})S_i(\tau). \tag{11}$$

Here, the term $D_{i,j}(t)/D_i$ represents the fraction of reduce tasks to place on DC j, which is proportional to the amount of input data that placed on it. Though such proportional task placement is usually not a wise choice, we believe that it can faithfully reflect the distribution of intermediate data, and is appropriate for characterizing the traffic cost incurred by a large number of query requests. To further reduce the cost, one can optimize the reduce task placement via existing solutions [6, 7], but please note that it is beyond the scope of this paper. Similarly, we also define the amount of intermediate data to be downloaded as $f^{2'}_{i,j}(\tau)$, which is calculated as follows

$$f^{2'}_{i,j}(\tau) = (D_i - D_{i,j}(t))\alpha_i \frac{D_{i,j}(t)}{D_i} S_i(\tau). \tag{12}$$

The link capacity constraints are shown as follows:

$$\sum_{i=1}^{N} \frac{f^2_{i,j}(\tau)}{\hbar} \le U^u_j, \forall j, \forall \tau, \tag{13}$$

$$\sum_{i=1}^{N} \frac{f^{2'}_{i,j}(\tau)}{\hbar} \le U^d_j, \forall j, \forall \tau. \tag{14}$$

Eq. (13) enforces the consumed bandwidth to not exceed the uplink capacity of DC j in each fine-grained time slot τ, while Eq. (14) is the downlink capacity constraint.

3.3 Characterizing the Cost-Throughput Tradeoff

3.3.1 Inter-DC traffic cost

The inter-DC traffic cost contains two parts: the traffic cost of input data movement and the traffic cost of intermediate data transmission. Note that we do not enforce any complicated traffic pricing model, e.g., 95th percentile pricing. The traffic cost is considered to be proportional to the volume of traffic. As such, the total inter-DC traffic cost in each coarse-grained time slot t is calculated as follows.

$$C(t) = \sum_{i=1}^{N}\sum_{j=1}^{M} f^1_{i,j}(t) + \sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N}\sum_{j=1}^{M} f^2_{i,j}(\tau). \tag{15}$$

Define $C(\tau) \triangleq \sum_{i=1}^{N}\sum_{j=1}^{M}\left(\frac{1}{T}f^1_{i,j}(t) + f^2_{i,j}(\tau)\right)$, and then we have $C(t) = \sum_{\tau=t}^{t+T-1} C(\tau)$. Here, $C(\tau)$ can be viewed as the total cost in each fine-grained time slot τ.

3.3.2 System throughput

For a large-scale GDA system, another important performance metrics is the overall system throughput in terms of the total number of query requests that can be simultaneously served. Specifically, in each fine-grained time slot

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing

τ, the system throughput is defined as the summation of served requests across all query types

$$S(\tau) = \sum_{i=1}^{N} S_i(\tau). \tag{16}$$

The cost-throughput tradeoff problem can now be formulated as the following stochastic problem **P1**:

$$\min_{D_{i,j}(t), S_i(\tau)} \quad \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{C(\tau) - \lambda \cdot S(\tau)\} \tag{17}$$

Subject to: *Eqs.* (1), (4), (5), (6), (7), (8), (9), (10), (13), (14).

Here, λ is a weight factor, representing how much we emphasize the throughput maximization. With such a weight factor, any desired trade-off point between the cost and performance can be achieved. We further observe that **P1** is a long-term optimization problem, where the current control decisions are coupled with future decisions. For example, current decisions may defer excessive query requests and hence block the service of future query requests. To solve such long-term optimization, the dynamic programming technique is a commonly used method [25], which, however, suffers from dimensionality issue and requires significant knowledge of the query request as well. Hence, we are motivated to take advantage of the Lyapunov framework [26] to design online control algorithms, without requiring prior knowledge of the query requests.

4 A Two-timescale Online Control Framework

In this section, we design a two-timescale online framework for solving the cost-throughput tradeoff problem. We start by transforming problem **P1** to problems that well fits the Lyapunov optimization technique, and then develop a two-timescale online algorithm which attempts to achieve a near-optimal solution without future statistics.

4.1 Transforming into Lyapunov Optimization

The key idea in the Lyapunov optimization technique is to transform a long-term optimization problem to many sub-problems, each of which can be solved in one time slot. To this end, we first introduce a set of delay-aware virtual queues, which can guarantee a maximal queuing delay l_{max} for each query request. Specifically, based on the technique of ϵ-persistent queue [27], we construct a group of delay-aware virtual queues $Y_i(t)$, $\forall i$, as shown in the following.

$$Y_i(t+1) = \max\{Y_i(t) - S_i(t) + \epsilon 1_{Q_i(t)>0}, 0\}, \forall i, \tag{18}$$

where $1_{Q_i(t)>0}$ is an indicator variable that is 1 if $Q_i(t) > 0$ and 0 otherwise. ϵ is a positive parameter, which is the key to ensure that $Y_i(t)$ grows whenever there are query requests in $Q_i(t)$ that has not been served. The following Lemma shows that the deferred query requests should be served within a worst-case delay l_{max} under any feasible algorithm.

Lemma 1. *If $Q(t) < Q_{max}$ and $Y_i(t) < Y_{max}$ are satisfied for any time slot t and any query type i, then any query request can be guaranteed with a maximal queuing delay l_{max} defined as follows*

$$l_{max} = \lceil (Q_{max} + Y_{max})/\epsilon \rceil. \tag{19}$$

Proof: At any time slot t, the newly arrived query requests $A_i(t) \geq 0, \forall i$. We focus on proving that the requests can be served before time $t + l_{max}$. If not, there is a contradiction. During time slots $\tau \in t+1, \ldots, t+l_{max}$, we know $Q_i(\tau) > 0$, otherwise $A_i(t)$ can be served before τ. Thus, $1_{Q_i(\tau)>0} = 1$. Based on Eq. (18), we have

$$Y_i(\tau+1) \geq Y_i(\tau) - S_i(\tau) + \epsilon, \forall i \tag{20}$$

Summing the above over $\tau \in t+1, \ldots, t+l_{max}$, we obtain

$$Y_i(t+l_{max}+1) - Y_i(t+1) \geq l_{max}\epsilon - \sum_{\tau=t+1}^{t+l_{max}} S_i(\tau), \forall i \tag{21}$$

Using the fact that $Y_i(t+1) \geq 0$ and $Y_i(t+l_{max}+1) \leq Y_{max}$, and rearranging the above inequality, we yield

$$\sum_{\tau=t+1}^{t+l_{max}} S_i(\tau) \geq l_{max}\epsilon - Y_{max}, \forall i \tag{22}$$

Since $Q_i(t+1) < Q_{max}$, we know that $A_i(t)$ will be served before $t + l_{max}$, whenever there are sufficient resource to serve at least Q_{max} amount of query requests during $\tau \in t+1, \ldots, t+l_{max}$. Since we have assumed that $A_i(t)$ is served before $t + Y_{max}$, we have $\sum_{\tau=t+1}^{t+l_{max}} S_i(\tau) \leq Q_{max}, \forall i$. Substituting this into the above inequality, we have

$$Q_{max} \geq l_{max}\epsilon - Y_{max} \tag{23}$$

Rearranging the terms, we get $l_{max} < (Q_{max} + Y_{max})/\epsilon$, contradicting the definition of l_{max} in Eq. (19). □

Now, we focus on transforming the original problem **P1** into Lyapunov optimization. Let $\mathbf{\Theta}(t)$ denote the concatenated vector of all queues, $\mathbf{\Theta}(t) = [Y_i(t), Q_i(t)]$. Then, we define the Lyapunov function as follows:

$$L(\mathbf{\Theta}(t)) = \frac{1}{2}\left(\sum_{i=1}^{N} Y_i^2(t) + \sum_{i=1}^{N} Q_i^2(t)\right) \tag{24}$$

This Lyapunov function quantitatively reflects the congestion of all queues [26], which should be persistently pushed towards a lower congestion state to keep the queue stabilities. Hence, we introduce T-slot conditional Lyapunov *drift* $\Delta_T(\mathbf{\Theta}(t))$, which is defined as follows

$$\Delta_T(\mathbf{\Theta}(t)) = \mathbb{E}\{L(\mathbf{\Theta}(t+T)) - L(\mathbf{\Theta}(t)) | \mathbf{\Theta}(t)\} \tag{25}$$

Based on the Lyapunov framework [26], it is the need to make decisions on $D_{i,j}(t)$, $S_i(\tau)$ to minimize the *drift-plus-penalty* term every T time slots. The *drift-plus-penalty* is defined as follows

$$\Delta_T(\mathbf{\Theta}(t)) + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1} (C(\tau) - \lambda S(\tau)) | \mathbf{\Theta}(t)\} \tag{26}$$

where $V(\geq 0)$ is a control parameter representing how much we emphasize the cost-performance tradeoff (Problem

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing

P1) compared to the system stability. The following theorem gives an upper bound for such *drift-plus-penalty* term.

Theorem 1. *Let $V\geq 0$, $\epsilon>0$, $T\geq 1$ and $t=kT, \tau\in[t, t+T-1]$. Assume that there exist certain peak levels for both arrival and service rates of query requests A_{max} and Q_{max}, such that $A_i(\tau)\leq A_{max}$ and $S_i(\tau)\leq Q_{max}, \forall i, \forall \tau$. Then, the drift-plus-penalty can be bounded as follows*

$$\begin{aligned}&\Delta_T(\mathbf{\Theta}(t)) + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}(C(\tau)-\lambda S(\tau))\,|\mathbf{\Theta}(t)\}\\ \leq& H_1 T + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}(C(\tau)-\lambda S(\tau))\,|\mathbf{\Theta}(t)\}\\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Y_i(\tau)\,(\epsilon - S_i(\tau))\,|\mathbf{\Theta}(t)\}\\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Q_i(\tau)\,(A_i(\tau)-S_i(\tau))\,|\mathbf{\Theta}(t)\}\end{aligned} \tag{27}$$

where $H_1 = \frac{N}{2}(2Q_{max}^2 + A_{max}^2 + \epsilon^2)$.

Proof: By using the fact that $(\max\{x,0\})^2 \leq x^2$, we have the following inequalities

$$\begin{aligned}\frac{1}{2}\sum_{i=1}^{N}(Y_i^2(\tau+1)-Y_i^2(\tau)) \leq& \frac{1}{2}\sum_{i=1}^{N} S_i^2(\tau) +\\ &\frac{1}{2}\sum_{i=1}^{N}(\epsilon 1_{Q_i(t)>0})^2 - \sum_{i=1}^{N} Y_i(\tau)\left(S_i(\tau)-\epsilon 1_{Q_i(t)>0}\right)\end{aligned}$$

$$\begin{aligned}\frac{1}{2}\sum_{i=1}^{N}(Q_i^2(\tau+1)-Q_i^2(\tau)) \leq& \frac{1}{2}\sum_{i=1}^{N}\left(S_i^2(\tau)+A_i^2(\tau)\right)\\ &-\sum_{i=1}^{N} Q_i(\tau)\,(S_i(\tau)-A_i(\tau))\end{aligned}$$

Summing these inequalities together and taking expectations over $Q_i(\tau)$ and $Y_i(\tau)$, we obtain the upper bound for 1-slot conditional Lyapunov drift $\Delta_1(\mathbf{\Theta}(t))$

$$\begin{aligned}\Delta_1(\mathbf{\Theta}(t)) \leq& H_1 + \sum_{i=1}^{N}\mathbb{E}\{Y_i(\tau)\,(\epsilon - S_i(\tau))\,|\mathbf{\Theta}(t)\}\\ &+\sum_{i=1}^{N}\mathbb{E}\{Q_i(\tau)\,(A_i(\tau)-S_i(\tau))\,|\mathbf{\Theta}(t)\}\end{aligned}$$

Adding the term $V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}(C(\tau)-\lambda S(\tau))\,|\mathbf{\Theta}(t)\}$ to both sides of the above inequality, over time slot $\tau\in[t, t+T-1]$, the theorem can then be proved. □

4.2 Two-timescale Online Control Algorithm

To design an online control algorithm, an intuitive approach is to minimize the upper bound of *drift-plus-penalty* term every T time slots. However, this needs to know the future concatenated queue backlog $\mathbf{\Theta}(t)=[Y_i(t), Q_i(t)]$ over time slot $\tau\in[t, t+T-1]$. $\mathbf{\Theta}(t)$ depends on the query request arrival process $A_i(\tau)$ and the decision $S_i(\tau)$, which, however, may not always be available. Due to the continuous variations of the system, we therefore approximate the near future queue backlog as the current value, i.e., $Y_i(\tau) = Y_i(t), Q_i(\tau) = Q_i(t)$ for all $t < \tau < t+T-1$. This significantly reduces the computational complexity of designing an online control algorithm. Nevertheless, such approximation leads to a "loosening" of the upper bound on the *drift-plus-penalty*, as shown in the following theorem. We will prove that our algorithm is robust against such approximation in Section 4.3.

Theorem 2. *Given **Theorem** 1 under approximation, the upper bound of the drift-plus-penalty can be loosened as follows*

$$\begin{aligned}&\Delta_T(t) + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}(C(\tau)-\lambda S(\tau))\,|\mathbf{\Theta}(t)\}\\ \leq& H_2 T + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}(C(\tau)-\lambda S(\tau))\,|\mathbf{\Theta}(t)\}\\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Y_i(t)\,(\epsilon - S_i(\tau))\,|\mathbf{\Theta}(t)\}\\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Q_i(t)\,(A_i(\tau)-S_i(\tau))\,|\mathbf{\Theta}(t)\}\end{aligned} \tag{28}$$

where $H_2 = H_1 + \frac{(T-1)N}{2}(\epsilon^2 + 2Q_{max}^2 + A_{max}^2)$.

Proof: According to Eq. (1) and (14), we have the following inequalities for any $\tau\in[t, t+T-1]$.

$$Q_i(t) - (\tau-t)Q_{max} \leq Q_i(\tau) \leq Q_i(t) + (\tau-t)A_{max} \tag{29}$$

$$Y_i(t) - (\tau-t)Q_{max} \leq Y_i(\tau) \leq Y_i(t) + (\tau-t)\epsilon \tag{30}$$

Applying the above inequalities in Eq. (27), we get

$$\begin{aligned}&\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Q_i(\tau)(A_i(\tau)-S_i(\tau))\\ \leq& \sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Q_i(t)(A_i(\tau)-S_i(\tau)) + \frac{T(T-1)N}{2}(A_{max}^2+Q_{max}^2)\end{aligned}$$

Similarly, we can also obtain

$$\begin{aligned}&\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Y_i(\tau)(\epsilon - S_i(\tau))\\ \leq& \sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Y_i(t)(\epsilon - S_i(\tau)) + \frac{T(T-1)N}{2}(\epsilon^2+Q_{max}^2)\end{aligned}$$

By defining $H_2 = H_1 + \frac{(T-1)N}{2}(\epsilon^2 + 2Q_{max}^2 + A_{max}^2)$ and using Eq. (27), the theorem can be proved. □

By substituting the definitions of $C(\tau)$ and $S(\tau)$ into Eq. (28), we get the following relaxed problem **P2**:

$$\begin{aligned}&\min V\mathbb{E}\{\sum_{i=1}^{N}\sum_{j=1}^{M}\max\{D_{i,j}(t-T)-D_{i,j}(t), 0\}|\mathbf{\Theta}(t)\}\\ &+V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N}\sum_{j=1}^{M} D_{i,j}(t)\alpha_i(1-\frac{D_{i,j}(t)}{D_i})S_i(\tau)|\mathbf{\Theta}(t)\}\\ &-\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N}(Y_i(t)+V\lambda)S_i(\tau)|\mathbf{\Theta}(t)\}\\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Q_i(t)\,(A_i(\tau)-S_i(\tau))\,|\mathbf{\Theta}(t)\}\end{aligned} \tag{31}$$

Subject to: *Eqs.* (1), (4), (5), (6), (7), (8), (9), (10), (13), (14).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing

Algorithm 1 2TGDA Online Control Algorithm

1: In the beginning of every coarse-grained time slot $t{=}kT, k{=}1,2,\ldots$, observing system state $Q_i(t)$ and $Y_i(t)$ ($\forall i$), determine the control decisions $D_{i,j}(t)$ to minimize the following problem **P3**:

$$\begin{aligned}\min_{D_{i,j}(t)}\ & V\mathbb{E}\{\sum_{i=1}^{N}\sum_{j=1}^{M}\max\{D_{i,j}(t-T)-D_{i,j}(t),0\}|\mathbf{\Theta}(t)\}\\ &+VT\mathbb{E}\{\sum_{i=1}^{N}\sum_{j=1}^{M}D_{i,j}(t)\alpha_i(1-\frac{D_{i,j}(t)}{D_i})Q_i(t)|\mathbf{\Theta}(t)\}\\ &\text{Subject to: } \textit{Eqs.}\ (4),(5),(6).\end{aligned} \tag{32}$$

2: At each fine-grained time slot $\tau\in[t,t{+}T{-}1]$, with the observed $Q_i(t)$, $Y_i(t)$, and the newly decisions $D_{i,j}(t)$, decide $S_i(\tau)$ to minimize the following problem **P4**:

$$\begin{aligned}\min_{S_i(\tau)}\ & \mathbb{E}\{\sum_{i=1}^{N}Q_i(t)\left(A_i(\tau)-S_i(\tau)\right)|\mathbf{\Theta}(t)\}\\ &-\mathbb{E}\{\sum_{i=1}^{N}(Y_i(t)+V\lambda)S_i(\tau)|\mathbf{\Theta}(t)\}\\ &\text{Subject to: } \textit{Eqs.}\ (1),(7),(8),(9),(10),(13),(14)\end{aligned} \tag{33}$$

3: Update the queue backlogs $Q_i(t)$, $Y_i(t)$ according to equalities (1) (18) and the newly determined decisions.

In the relaxed problem **P2**, the decision variables, $D_{i,j}(t)$ and $S_i(\tau)$, are still coupled with each other in the term of $V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N}\sum_{j=1}^{M}D_{i,j}(t)\alpha_i(1-\frac{D_{i,j}(t)}{D_i})s_i(\tau)|\mathbf{\Theta}(t)\}$. In our online algorithm, we replace $S_i(\tau)$ with $Q_i(t)$ in this term, so as to enable the decision-making on two timescales. Our online two-timescale 2TGDA control algorithm is illustrated in Algorithm 1. Specifically, at the beginning of each coarse-grained time slot t, 2TGDA decides how much amount of input data to be placed on each DC for each query type, by solving problem **P3**. Then, at the beginning of each fine-grained time slot $\tau\in[t,t{+}T{-}1]$, it determines how many query requests can be executed simultaneously, by solving problem **P4**. At the end of each fine-grained time slot, it updates the queue backlogs.

Our 2TGDA is computationally efficient. The dominate operation of it is to solve a convex program (i.e., **P3**) and a linear program (i.e., **P4**). While both programs can be solved with classical optimization approaches, e.g., interior point method [28, 29], which can have a low computational complexity (usually polynomial time). We denote $CP(x,y)$ ($LP(x,y)$) as the time complexity of solving a convex program (linear program) with x variables and y constraints. Then, 2TGDA can run in $O(O(CP(O(MN),O(MN))+O(T\cdot LP(O(N),O(N{+}M)))$. An additional source of overhead is the time to gather input information and distribute new decisions at the beginning of each coarse time slot t. This messaging delay could be about tens of milliseconds as similar to [30], which can be negligible as compared to the length of a coarse time slot.

We highlight 2TGDA's two interesting properties. *First*, 2TGDA enables flexible tradeoff point between traffic cost and system throughput by tuning the weight factor λ, while also guarantees a maximal delay and system stability by appropriately tuning the control parameter V and delay control parameter ϵ. *Second*, 2TGDA may choose not to serve query requests in a particular time slot, even if the queue backlog $Q_i(t)>0$, due to a high traffic cost. This may introduce additional queuing delay but can reduce the inter-DC traffic cost. 2TGDA provides opportunities for traffic cost savings because the accumulate intermediate data of continuous arriving query requests may be larger than the input-data in size. While the distribution of input-data carries over to the distribution of intermediate data. So, this motivates us to re-distribute the input-data ahead of time to reduce the traffic cost.

4.3 Performance analysis

We now analyze the performance achieved by 2TGDA. Let $O(\tau)=C(\tau)-\lambda S(\tau)$, and let O^{opt} denote the optimal objective value of the original problem **P1**. The following theorem shows the gap between the result achieved by our 2TGDA and the optimal solution

Theorem 3 (Performance Bound Analysis). *Assume that there exist certain perk level for the ration of intermediate data to input data α_{max}, such that $\alpha_i\le\alpha_{max}$. Then, for any $V{>}0$, 2TGDA can achieve the following performance guarantee:*

$$\lim_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\mathbb{E}\{O(\tau)|\mathbf{\Theta}(t)\}\le\frac{H_2}{V}+O^{opt}+D\alpha_{max}Q_{max} \tag{34}$$

$$\bar{\Theta}\le\frac{H_2+VO^{opt}+VD\alpha_{max}Q_{max}}{\xi} \tag{35}$$

where $\xi>0$ is a positive parameter and $\bar{\Theta}$ is the time-averaged queue length that is computed as $\bar{\Theta}=\lim_{t\to\infty}\frac{1}{K}\sum_{k=0}^{K-1}\sum_{i=1}^{N}(Q_i(kT)+Y_i(kT))$.

Proof: It has been proved in [26] that there exists at least one randomized stationary control policy Π^{opt} that chooses feasible control decisions $D_{i,j}(t)$ every T time slots, and chooses $S_i(\tau)$ every time slot, and yields the following steady state values:

$$\sum_{\tau=t}^{t+T-1}O(\tau)=TO^{opt};\epsilon=S_i(\tau)-\mu;A_i(\tau)=S_i(\tau)-\nu \tag{36}$$

Since 2TGDA solves **P3** and **P4** every T time slots, integrating Eq. 36 into Eq. (28), using the fact $Q_i(t)\le Q_{max},\alpha_i(t)\le\alpha_{max}$ and defining $D=\sum_{j=1}^{M}\sum_{i=1}^{N}D_{i,j}(t)$, we get

$$\begin{aligned}&\Delta_T(t)+V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\left(C(\tau)-\lambda S(\tau)\right)|\mathbf{\Theta}(t)\}\\ &\le H_2T+VTO^{opt}\\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{j=1}^{M}\sum_{i=1}^{N}VD_{i,j}(t)\alpha_i(1-\frac{D_{i,j}(t)}{D_i})(Q_i(t)-S_i(\tau))\}\\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N}Y_i(t)\left(\epsilon-S_i(\tau)\right)+Q_i(t)\left(A_i(\tau)-S_i(\tau)\right)|\mathbf{\Theta}(t)\}\\ &\le H_2T+VTO^{opt}+VTD\alpha_{max}Q_{max}-T\sum_{i=1}^{N}(\mu Y_i(t)+\nu Q_i(t))\end{aligned} \tag{37}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing

Rearranging the terms, we get

$$V\mathbb{E}\{\sum_{\tau=t}^{t+T-1} O(\tau)|\mathbf{\Theta}(t)\}\leq H_2T+VTO^{opt}+VTD\alpha_{max}Q_{max} \quad (38)$$

Summing the above over $t=kT, k=0,1,2,\ldots,K-1$, and dividing both sides by KTV, we have:

$$\frac{1}{KT}\mathbb{E}\{\sum_{\tau=0}^{KT-1} O(\tau)|\mathbf{\Theta}(t)\}\leq\frac{H_2}{V}+O^{opt}+D\alpha_{max}Q_{max} \quad (39)$$

Taking limit as $K\to\infty$, we can yield Eq. (34). Now, defining $\xi=\min\{\mu,\nu\}$, we also obtain the following inequality by rearranging the terms of Eq. (37).

$$T\xi\sum_{i=1}^{N} Y_i(t)+Q_i(t)\leq H_2T+VTO^{opt}+VTD\alpha_{max}Q_{max} \quad (40)$$

Summing the above over $t=kT, k=0,1,2,\ldots,K-1$, dividing both sides by KTV, and taking limit as $K\to\infty$, we can yield Eq. (35). □

Recall that 2TGDA approximates the future concatenated queue backlogs $\mathbf{\Theta}(t)=[Y_i(t),Q_i(t)]$ as its current value. So, what happens when 2TGDA makes decisions based on queue backlog estimates $\hat{\mathbf{\Theta}}(t)$ that differs from the actual backlogs? The following theorem shows that our 2TGDA algorithm is robust against the queue backlogs estimation errors.

Theorem 4 (Robustness of 2TGDA). *Suppose that there exists a constant σ, such that at all time t, the estimated backlog satisfy $|\hat{Y}_i(t)-Y_i(t)|\leq\sigma$ and $|\hat{Q}_i(t)-Q_i(t)|\leq\sigma$. Then, under the 2TGDA algorithm, we have*

$$\begin{aligned} O^{Robust} &\triangleq \lim_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\mathbb{E}\{O(\tau)|\mathbf{\Theta}(t)\} \\ &\leq \frac{H_3}{V}+O^{opt}+D\alpha_{max}Q_{max} \end{aligned} \quad (41)$$

where $H_3=H_2+T\sigma(2VD\alpha_{max}+N(\epsilon+2Q_{max}+A_{max}))$.

Proof: Denote $e^Y(t)=\hat{Y}_i(t)-Y_i(t)$ and $e^Q(t)=\hat{Q}_i(t)-Q_i(t)$. $O(\hat{\Theta}(t))$ can then be calculated as:

$$\begin{aligned} O(\hat{\mathbf{\Theta}}(t)) &= O(\mathbf{\Theta}(t))+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} e^Y(t)\left(\epsilon-S_i(\tau)\right)|\mathbf{\Theta}(t)\} \\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N}\sum_{j=1}^{M} e^Q(t)VD_{i,j}(t)\alpha_i(1-\frac{D_{i,j}(t)}{D_i})|\hat{\mathbf{\Theta}}(t)\} \\ &+\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} e^Q(t)\left(A_i(\tau)-S_i(\tau)\right)|\mathbf{\Theta}(t)\} \end{aligned}$$

Using the fact that $|a-b|\leq|a|+|b|$, we have $|1-\frac{D_{i,j}(t)}{D_i}|\leq 1+\frac{D_{i,j}(t)}{D_i}\leq 2$, $|\epsilon-S_i(\tau)|\leq\epsilon+Q_{max}$, $|A_i(\tau)-S_i(\tau)|\leq A_{max}+Q_{max}$. Denote the minimum value of $O(\hat{\mathbf{\Theta}}(t))$ and $O(\mathbf{\Theta}(t))$ as $\hat{O}$ and O_*, respectively. Then, we have

$$\hat{O}\leq O_*+T\sigma(2VD\alpha_{max}+N(\epsilon+2Q_{max}+A_{max})) \quad (42)$$

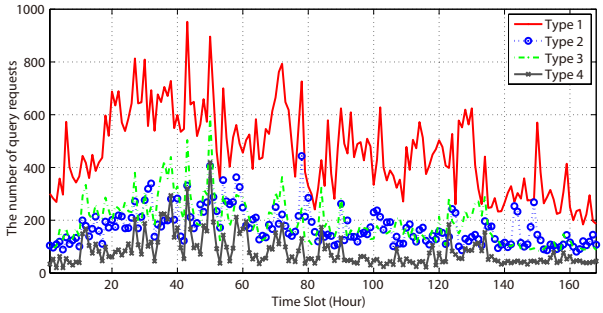


Fig. 3. 7-day real world traces from [31].

This shows that Eq. (28) holds with $Q_i(t)$, $Y_i(t)$, replaced by $\hat{Q}_i(t)$, $\hat{Y}_i(t)$, and H_2 replaced by $H_3=H_2+T\sigma(2VD\alpha_{max}+N(\epsilon+2Q_{max}+A_{max}))$. The rest of the proof is similar to the proof of Eq. (34) in theorem 3. □

5 Performance Evaluation

We evaluate the performance of our proposed algorithm through large-scale trace-driven simulation.

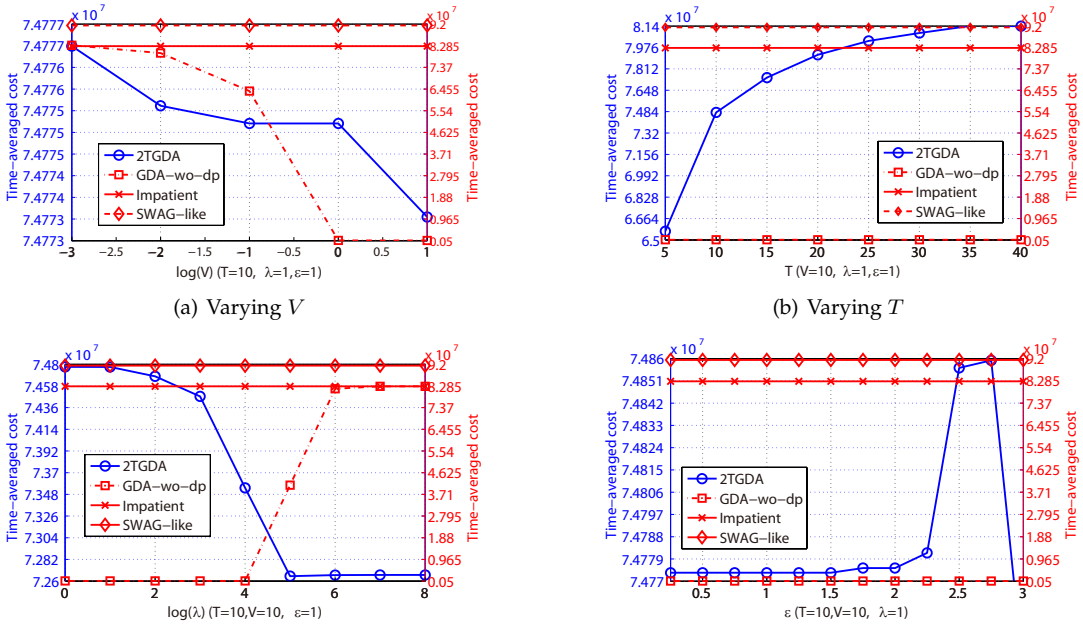
5.1 Experiment setup

Datasets: To simulate the arrival patterns of query requests, we use Google cluster traces [31], which contain the information about job submissions during a period of 29 days. Each job is viewed as a query request. These jobs are divided into four types, as the *scheduling class* information in this trace indicates the type of the job and its value ranges from 0 to 3. Specifically, we extract the information of jobs in a 7-day duration. The extracted traces contain 168 time slots, with each time slot being 1 hour. Fig. 3 plots the number of requests every 1-hour, for all query types.

Parameters settings: We consider a GDA system with 30 DCs, which is a common size in typical service companies [6]. In this 30-DC setup, the uplink and downlink bandwidths of each DC are all randomly distributed within $[1,10]$Gbps. Initially, for each query type, the amount of input data stored on each DC is randomly generated as a uniform distribution within $[1,10]$Gb. The ratios of intermediate data to input data for the four query types (e.g., $\alpha_i,\forall i$) are set to be 0.5, 0.7, 0.9, 1.1, respectively. We conduct sensitivity analysis on critical parameters, e.g., T, V, λ and ϵ, to evaluate their impact on the performance of our algorithm, in terms of the time-averaged cost, the time-averaged throughput, and the maximum delay. Specifically, these parameters are varied as follows: $T\in[5,40]$, $V\in[10^{-3},10]$, $\lambda\in[1,10^8]$, $\epsilon\in[0.25,3]$.

Compared algorithms: We compare our *2TGDA* with the following three algorithms. The first algorithm aims at minimizing the traffic cost caused only by intermediate data while maximizing the throughput, which is also designed based on the Lyapunov optimization techniques. It is different from our *2TGDA* as it ignores the input data placement, and hence we refer it as "*GDA-wo-dp*". The second algorithm is an online algorithm that always schedules query requests immediately at each time slot to maximize throughput, regardless of the inter-DC traffic cost, which is referred to as "*Impatient*". The last algorithm schedules query requests

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing



(a) Varying V (b) Varying T

(c) Varying λ (d) Varying ϵ

Fig. 4. The performance of time-averaged cost under different values of (a) control parameter V, (b) the number T of fine-grained time slots in each coarse-grained time slot, (c) weight factor λ and (d) delay-associated parameter ϵ.

greedily based on their estimated finish time. For each query request, its estimated finish time contains two parts: the completion time when it exclusively occupies all links and the waiting time of its flow transfers on relevant links. This algorithm essentially takes the existing workload at local links into consideration and is conceptually equivalent to the Workload-Aware Greedy Scheduling (SWAG) [16]. Hence, we denote it as "*SWAG-like*". We evaluate the above algorithms with the following three performance metrics by default: time-averaged cost, time-averaged throughput, and maximum queuing delay.

5.2 Experiment results

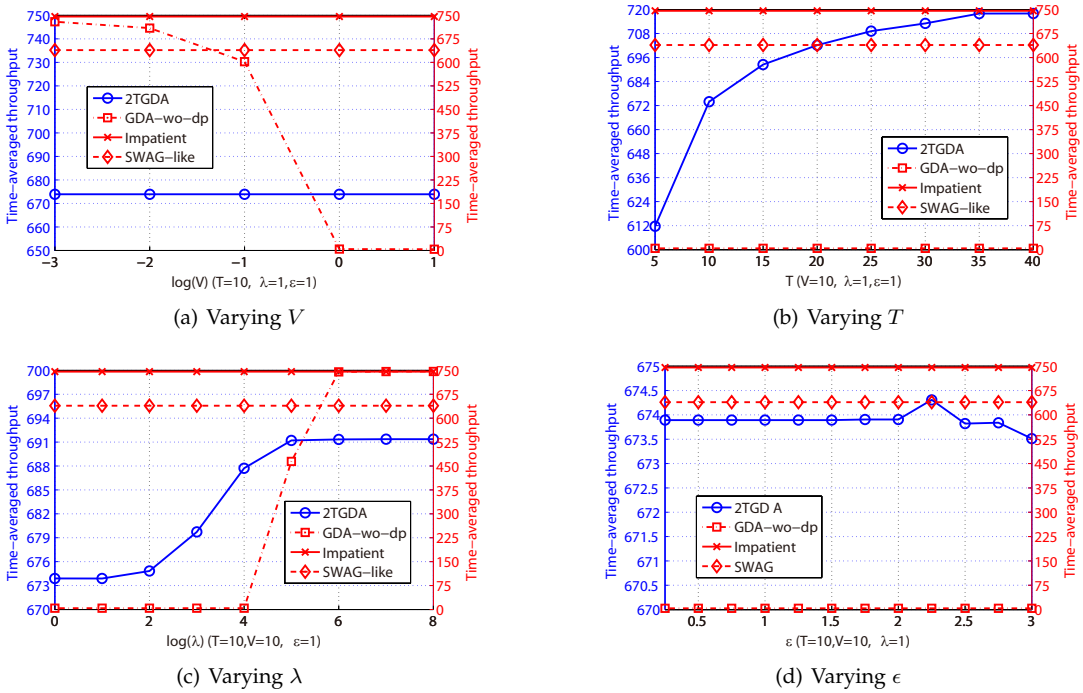
5.2.1 Inter-DC traffic cost

The inter-DC traffic cost is one of the most important performance metrics for the GDA system. To have a comprehensive understanding, we study the impacts of different parameters (e.g., V, T, λ, ϵ) on the time-averaged cost, as shown in Fig. 4. Note that when evaluating the impact of each parameter on the inter-DC traffic cost, we fix the values of other parameters to be constants. From Fig. 4, we have the following observations. *First*, no matter how the parameters change, the time-averaged cost achieved by both *Impatient* and *SWAG-like* will never change. This is because these two algorithms immediately schedule the query requests at each time slot, as long as meeting the link bandwidth capacity. *Second*, the time-averaged cost achieved by *GDA-wo-dp* decreases significantly with the increasing of V, while increases as λ increases. One reason is that *GDA-wo-dp* uses Lyapunov optimization techniques [23]. Another reason is that *GDA-wo-dp* also studies the cost-performance problem, and λ emphasizes the importance of the system throughput maximization. Hence, this eventually enforces *GDA-wo-dp* to achieve a higher cost with a larger value of λ. As *GDA-wo-dp* ignores the input data placement, it always maintains a stable cost regardless of the change of long-term time slot T. Furthermore, the time-averaged cost achieved by *GDA-wo-dp* will never change, as the increase of ϵ. One reason may be that ϵ is not sufficiently large to make the delay-aware queue backlog $Y_i(t)$ to be changed across all time slots. *Third*, the time-averaged cost achieved by our *2TGDA* has different trends with the increasing of the parameters. More precisely, it decreases as the control parameter V increases. This directly confirms the Theorem 3. On the contrary, it increases with the increase of T, which verifies the effectiveness of input data replacement in reducing the inter-DC traffic cost. An interesting observation is that the time-averaged cost achieved by our *2TGDA* algorithm decreases slightly as the increasing of λ, which is the weight factor indicating the importance of system throughput maximization. The main reason is that our *2TGDA* algorithm tactfully considers the input-data placement, saving the traffic cost for intermediate data. Moreover, more cost can be saved if there are more concurrent query requests. We further observe that the time-averaged cost of *2TGDA* maintains at a low value with small ϵ, and has a higher value when $\epsilon = 2.5$ or 2.75. The time-averaged cost of our *2TGDA* is higher than that of *GDA-wo-dp* for some of the cases, but is always lower than that of *Impatient* and *SWAG-like*. However, our *2TGDA* can reduce significant inter-DC traffic cost by choosing a higher value of V (e.g., $V = 10$), a lower value of T (e.g., $T = 5$), a higher value of λ (e.g., $\lambda = 100000$), and a lower value of ϵ (e.g., $\epsilon = 0.25$).

5.2.2 System throughput

Another important performance metric is the overall system throughput in terms of the number of query requests that can be simultaneously served. We can also observe that the time-averaged throughput achieved by both *Impatient* and *SWAG-like* always maintains a stable value, irrespective of the change of any parameters, e.g., V, T, λ and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing



(a) Varying V (b) Varying T

(c) Varying λ (d) Varying ϵ

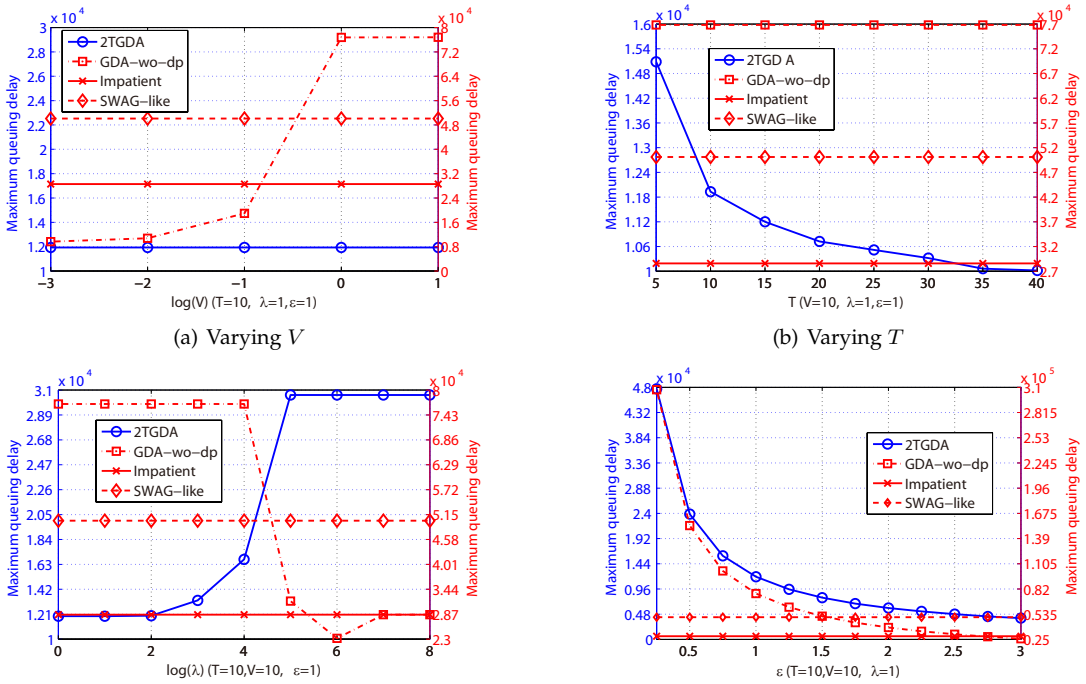
Fig. 5. The performance of time-averaged throughput under different values of (a) control parameter V, (b) the number T of fine-grained time slots in each coarse-grained time slot, (c) weight factor λ and (d) delay-associated parameter ϵ.

ϵ. For the *GDA-wo-dp* algorithm, it achieves a decreasing throughput as V increases and an increasing throughput as λ increases. *GDA-wo-dp* maintains a stable value of time-averaged throughput, regardless of the change of T and ϵ. For our *2TGDA* algorithm, the time-averaged throughput does not change with the increase of the control parameter V. The root reason is that the increase of V will not influence the decision-making process of problem **P4**. With the increase of T, the time-averaged throughput achieved by *2TGDA* also increases. This is because the number of time slots with $S_i(\tau) = 0$ is less, which indirectly improves the time-averaged throughput. *2TGDA* achieves an increasing time-averaged throughput with the increasing of λ since λ is exactly the weight factor emphasizing the importance of system throughput maximization. We further observe that *2TGDA* maintains a relatively stable time-averaged throughput as ϵ increases. This is because that ϵ decides the value of $Y_i(t)$, while $Y_i(t)$ does not affect the decision $S_i(t)$ too much in problem **P4**. In general, the time-averaged throughput of *2TGDA* is between that of *GDA-wo-dp* (*SWAG-like*) and *Impatient*. Moreover, *2TGDA* can achieve acceptable system throughput by choosing appropriate values of critical factors, e.g., $V = 10, T = 40, \lambda = 100000, \epsilon = 0.25$.

5.2.3 Queuing delay

Last but not the least, we investigate the performance of different algorithms on queuing delay. Recall that the our model can guarantee a maximum queuing delay l_{max} for each query request, and l_{max} is given in lemma 1. We therefore use the maximum queuing delay l_{max} as the performance metric. Specifically, we plot the impact of different parameters on the maximum queuing delay, as shown in Fig. 6. It can easily be checked that both *Impatient* and *SWAG-like* can maintain a stable values of the maximum queuing delay, regardless the changes of any parameters, e.g., V, T, λ and ϵ. For *GDA-wo-dp*, the maximum queuing delay l_{max} increases as V increases, which directly verifies an decreasing of time-averaged throughput in Fig. 5(a). Furthermore, *GDA-wo-dp* achieves a stable value of l_{max} regardless of the change of T, and has a decreasing value of l_{max} as λ or ϵ increases. This is due to the following reasons: *1) GDA-wo-dp* ignores the data placement in each coarse-grained time slot, *2)* a larger λ leads to a larger throughput, and *3)* a larger ϵ leads to a smaller l_{max} based on the definition in Eq. (19). For our *2TGDA* algorithm, the maximum queuing delay is maintained at a stable value with the increasing of V because of the unchanged throughput. Furthermore, the maximum queuing delay achieved by *2TGDA* decreases as the value of T or ϵ increases. The root reason can be derived from the throughput achieved by *2TGDA* and the definition of the maximum queuing delay l_{max}. Note that l_{max} achieved by *2TGDA* increases as λ increases, while the throughput achieved by *2TGDA* also increases with the increasing of λ. One of reason may be that a higher value of λ always leads to higher Q_{max} or Y_{max}, but leads to a lower value of the average queue backlog across all time slots and all query types. In most cases, the maximum queuing delay guaranteed by our *2TGDA* falls between *GDA-wo-dp* and *Impatient* (*SWAG-like*). Furthermore, with appropriate values of $V = 10, T = 40, \lambda = 1, \epsilon = 3$, *2TGDA* can achieve a considerable low delay for each query request. One may further wonder why *SWAG-like* achieves inferior performance on maximum queuing delay, since it is mindful of the finish time of query requests. The underlying reason is that the finish time estimated by *SWAG-like* considers only the flow transmission time and the waiting time in each link, without the queuing time at the centralized scheduler.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing



(a) Varying V (b) Varying T

(c) Varying λ (d) Varying ϵ

Fig. 6. The performance of maximum queuing delay l_{max} under different values of (a) control parameter V, (b) the number T of fine-grained time slots in each coarse-grained time slot, (c) weight factor λ and (d) delay-associated parameter ϵ

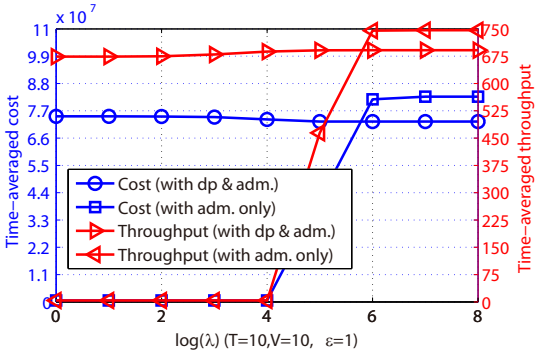


Fig. 7. Cost-throughput tradeoff achieved by our algorithm.

5.2.4 *Cost-throughput tradeoff*

Fig. 7 shows the performance on trading off the cost minimization and throughput maximization for our algorithm. We observe that when input data placement and query request admission control are both enabled, our algorithm can maintain a slightly increasing throughput and a slightly decreasing cost, with the increasing of λ. This implies that our algorithm can achieve low cost and high throughput simultaneously with an appropriate λ. One may question why the cost can decrease as λ increases since λ is a weight factor for throughput maximization. The root reason lies in the input data placement optimization, and more requests enable this optimization to save more cost. On the contrary, if we disable the input data placement, we find that the cost can hike significantly with the increase of λ, while the throughput can keep increasing. So, by choosing an appropriate value of λ, our algorithm can well tradeoff the cost minimization and the throughput maximization.

6 Discussion

Considering delay or query completion time: Our work only considers queuing delay for each query request (or job) at the master DC, instead of the total completion time of the job. In fact, a job's completion time is composed of the waiting time (at both the master DC and the worker DCs) as well as the job service time [16]. To consider such completion time in our work, a simple approach could do the following. Let $W_1(\tau)$ denote the waiting time of a query job at the master DC, which could be expressed as some function of the queue backlogs $\sum_i Q_i(\tau)$. Let $W_2(\tau)$ indicate the waiting time of every job at worker DCs. $W_2(\tau)$ can be computed as $\max_j(\max(\mathcal{L}_j^u(\tau)/U_j^u, \mathcal{L}_j^d(\tau)/U_j^d))$ if each DC uses FIFO to schedule its flow transfers, where $\mathcal{L}_j^u(\tau)$ and $\mathcal{L}_j^d(\tau)$ indicate the accumulated traffic load to be uploaded and downloaded by DC j at time τ, respectively. Let $W_3^i(\tau)$ represent the minimum transfer time required for the flows of an i-th type job at time τ, which can be calculated by $\max_j(\max(D_{i,j}(t)\alpha_i(1-D_{i,j}(t)/D_i)/U_j^u, (D_i - D_{i,j}(t))\alpha_i D_{i,j}(t)/D_i/U_j^d))$. Summing up $W_1(\tau)$, $W_2(\tau)$ and $W_3^i(\tau)$ and enforcing the summation to not exceed a given budget (e.g., C_{max}) at each time slot τ, one can incorporate a maximum completion time constraint for each query job into our problem formulation. The last thing is to redesign queuing dynamics to solve the problem with the Lyapunov optimization techniques. We leave it to future work.

Handling hybrid cloud scenarios: Our work is only applicable to a private cloud or public cloud, not a hybrid environment with a mix of private and public clouds. Actually, in a hybrid cloud scenario, some regulatory and privacy concerns might forbid the input data movements for specific DCs [6, 32]. To handle this scenario, one simple approach could do the following. Let $v_{i,j}$ ($\geq$ 0) denote

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2020.2994195, IEEE Transactions on Cloud Computing

the volume of input data must be kept at DC j for i-th query type. The value of $v_{i,j}$ can be specified by the operator based on its regulatory concerns, and a value of zero implies that there is no constraint on the relevant input data movement. Given the definition above, one can add the following constraint: $D_{i,j}(t) \geq v_{i,j}, \forall t, \forall i, \forall j$, to the problem formulation in Sec. 3.3. Then, one can design heuristic to avoid a DC being violated its input data constraint, to solve the problem. Alternatively, one can reuse the basic framework of our 2TGDA algorithm, and replace Step 1 by formulating and solving a new problem **P3** that incorporates the data constraint above.

Enhancing theoretical contribution: Our work is based on the standard Lyapunov optimization technique, as we believe it can solve our problem efficiently, and enhancing the Lyapunov optimization tool is beyond the scope of this paper. But of course, the standard Lyapunov technique is not without problem. For example, it relies on solving the decomposed sub-problems optimally. If the sub-problems happen to be NP-hard or large in scale and cannot be solved optimally, it is unclear whether the standard Lyapunov technique can still provide a theoretical performance guarantee. One can assume an error in minimizing (or maximizing) the sub-problem and try to analyze if the Lyapunov optimization technique can tolerate such error and provide a performance guarantee with respect to the original (perhaps stochastic) problem. As such, the theoretical contribution of using the Lyapunov optimization technique could be somehow strengthened. We leave it as an open problem.

Implementation issues: The proposed algorithm is evaluated through large-scale trace-driven simulations only, without real system-based implementation over real geo-distributed DCs (e.g., Amazon EC2 DCs). The reasons are as follows. First, existing GDA systems (e.g., MapReduce and Spark) largely lack interfaces to dynamically or arbitrarily split and migrate the input data on each DC, and developing such interfaces from scratch requires substantial efforts. Second, our model and algorithm require all query jobs or requests must be completed before performing the input data movement; otherwise, the system may be crashed. To meet this requirement, the operator may need to understand the job workload beforehand and precisely determine the scale of a time slot. Last but not least, since the problem we solved is a long-term optimization, the high cost of renting computational instances for a long time prohibits us from building a testbed on commodity cloud platforms (e.g., Google Cloud Engine or Amazon EC2). In view of the reasons above, we leave the real testbed implementation to future work.

7 Related Work

In this section, we will summarize the most related work in geo-distributed data analytics, which can be divided into two categories based on their objectives. They mainly focus on reducing the amount of inter-DC traffic or shortening job completion time.

To reduce the bandwidth costs, Vulimiri et al. solve an integer linear program to optimize the query execution plan and aggressively cache the results of prior queries for the subsequent queries [7]. However, they do not consider the constraints on the data movement, and sometimes the algorithm they used is slow. Pixida is a scheduler that takes advantage of the graph partition method to minimize the data transfers among datacenters [13]. WANalytics is an extended version of Pixida, and also uses an efficient cache mechanism to reduce data transfer among datacenters [14]. It will cause overhead for caching and computations within a datacenter. Ariel et al. propose JetStream, which uses aggregation and degradation for reducing the traffic cost in stream processing across wide area networks [4]. Relative to JetStream, Heintz et al. study a tradeoff problem between timeliness and accuracy in geo-distributed streaming analytics [33]. Nevertheless, both of them care about the bandwidth constraints by trading streaming query accuracy while our work preserves the data fidelity. Though those solutions can somehow reduce the inter-DC bandwidth cost, they do not consider the size relationship between input data and the intermediate data generated by many queries. Moreover, they do not consider any performance issues related to the geo-distributed data analytics queries, e.g., the job completion time or system throughput (the number of queries that can be served).

To reduce the job completion time, Iridium optimizes the placement of both the input data and reduce task [6]. Nevertheless, Iridium cannot guarantee the input data movement among datacenters to be completed before the query arrives. Wang et al. propose Turbo to dynamically adjust the execution plan for a geo-distributed analytical query by predicting the time cost of a plan with machine learning [34]. Turbo is a single-query solution, which ignores the scheduling of multiple queries. Hung et al. propose SWAG, which optimizes the average completion time of jobs running across multiple geo-distributed DCs by greedily scheduling the job with the shortest estimated finish time [16]. The estimated finish time of each job considers the remaining size/progress of the job and the queue length among all DCs. Nonetheless, SWAG is not aware of the optimization of data placement. Convolbo et al. propose GeoDis to jointly consider data replication and task migration to minimize the makespan of jobs [35]. Though SWAG and GeoDis show very promising performance improvement in makespan minimization, they both ignore the inter-DC traffic minimization and also make no attempt to improve the number of successfully served jobs/queries per unit time. Hwang et al. propose to replicate stream processing operators, with the aim of achieving fast and reliable processing for geo-distributed analytics [36]. Pietzuch et al. investigate to design a layer between a stream-processing system and the physical network that manages operator placement, with a series of objectives, e.g., improving network utilization and providing low stream latency [37]. They both focus on optimizing the operator placement and are orthogonal to our work. CLARIENT is a recently proposed solution for reducing the query completion time, which uses heuristics to jointly selects the placements and schedules of tasks [17]. CLARIENT is applicable to both single-query and multi-query scenarios. However, CLARIENT is unaware of the inter-DC traffic generated by analytics queries. It may end up incurring massive amounts of inter-DC traffic that uses the entire WAN budget even if the improvement in query completion time is negligible. Moreover, despite the

unawareness of inter-DC traffic, CLARIENT's assumption on simultaneous query arrivals is also unrealistic.

There also exist other works that try to reduce job or query completion time while being mindful of reducing inter-DC traffic (cost). For example, Iridium [6] uses a pre-configured WAN (Wide Area Network) usage budget to prevent causing too much bandwidth cost. Practically, it is hard to set an optimal budget as we have no insight into the amount of traffic workload in WAN. Jayalath et al. propose G-MR, which considers both job completion time and cost when executing a sequence of jobs [15]. Hsieh et al. propose Gaia [11] to speed up machine learning training over WAN. Its key idea is to transfer only the gradients that are significant enough for model refining, so as to reduce communication overhead over WAN while retaining the accuracy of a machine learning algorithm. Heintz et al. [38] focus on optimizing the tradeoff between delay and traffic cost for windowed grouped aggregation in geo-distributed streaming analytics, while Kumar et al. [39] concern such delay-cost tradeoff for continuous aggregation. Our work is different from both these works. First, we consider the optimization of geo-distributed analytics over batch data rather than streaming data. Second, we focus on the throughput-cost tradeoff instead of delay-cost tradeoff.

Besides, a growing body of recent work has leveraged Lyapunov optimization techniques to solve a variety of problems in DC networks. For example, Liu et al. [40] and Yao et al. [24] propose to use Lyapunov optimization techniques to manage the power or energy consumption efficiently in DC networks. Chen et al. [41] develop a Lyapunov optimization based transport to schedule a mixture of deadline/non-deadline flows in DC networks. Li et al. [42] leverage Lyapunov optimization framework to explore the free time slots for inter-DC traffic transmission. Our work is different, as we use Lyapunov techniques to derive the joint decisions of input data placement and query request admission for the GDA system, to minimize the inter-DC traffic cost and improve the system throughput.

8 Conclusions

This paper studies the problem of jointly considering input data placement and admission control to minimize the inter-DC traffic cost and maximize throughput (i.e., the number of simultaneously served requests) for a GDA system. We take advantage of Lyapunov optimization to design a two-timescale online control framework. Without requiring a prior knowledge of subsequent query requests, *2TGDA* makes online decisions on input data placement at the beginning of each coarse-grained time slot, and the query request admission at the beginning of each fine-grained time slot. We conduct rigorous theoretical analysis to demonstrate the optimality and robustness of our proposed algorithm. Finally, we use trace-driven simulation to verify that our algorithm is effective in arbitrating the cost-performance tradeoff and guaranteeing the queuing delay for query requests.

Acknowledgment

This work was supported partly by the NSFC General Technology Basic Research Joint Funds under Grant U1836214; the State Key Program of National Natural Science of China under Grants 61832013; the Artificial Intelligence Science and Technology Major Project of Tianjin under Grant 18ZXZNGX00190; the National Key R&D Program of China under Grant 2019QY1302; the NSFC under Grant 61672379; the National Key R&D Program of China under Grant 2019YFB2102404; the NSFC-Guangdong Joint Funds under Grant U1701263; the Natural Science Foundation of Tianjin under Grant 18ZXZNGX00040; the National Key R&D Program of China under Grant 2018YFB1004700; the NSFC under Grants 61872265, 61672131; the Key research and Development Program for Guangdong Province 2019B010136001; the NSFC under Grants 61772112, U1811463; the Science Innovation Foundation of Dalian under Grant 2019J12GX037.

References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proc. of ACM SIGCOMM*, 2013.

[2] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proc. of ACM SIGCOMM*, 2013.

[3] A. Gupta, F. Yang, J. Govig, A. Kirsch, K. Chan, K. Lai, S. Wu, S. G. Dhoot, A. R. Kumar, A. Agiwal *et al.*, "Mesa: Geo-replicated, near real-time, scalable data warehousing," in *Proceedings of the VLDB*, 2014.

[4] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and degradation in jetstream: Streaming analytics in the wide area," in *Proceedings of USENIX NSDI*, 2014.

[5] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proceedings of ACM SOSP*, 2013.

[6] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proceedings of ACM SIGCOMM*, 2015.

[7] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proceedings of USENIX NSDI*, 2015.

[8] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in *Proc. of ACM SIGMOD*, 2010.

[9] G. Lee, J. Lin, C. Liu, A. Lorek, and D. Ryaboy, "The unified logging infrastructure for data analytics at twitter," *Proc. of the VLDB Endowment*, vol. 5, no. 12, pp. 1771–1780, 2012.

[10] "Yahoo cross data-center data movement," 2010. [Online]. Available: http://yhoo.it/1nPRImNl

[11] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching lan speeds," in *Proc. of USENIX NSDI*, 2017.

[12] R. Ananthanarayanan, V. Basker, S. Das, A. Gupta, H. Jiang, T. Qiu, A. Reznichenko, D. Ryabkov, M. Singh, and S. Venkataraman, "Photon: fault-tolerant and scalable joining of continuous data streams," in *Proceedings of ACM SIGMOD*, 2013.

[13] K. K. M. M. N. Preguiça and R. Rodrigues, "Pixida: Optimizing data parallel jobs in bandwidth-skewed environments," in *Proceedings of VLDB Endowment*, 2015.

[14] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese, "Wanalytics: Analytics for a geo-distributed data-intensive world." in *CIDR*, 2015.

[15] C. Jayalath, J. Stephen, and P. Eugster, "From the cloud to the atmosphere: running mapreduce across data centers," *IEEE Transactions on Computers (TC)*, vol. 63, no. 1, pp. 74–87, 2014.

[16] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 111–124.

[17] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "Clarinet: Wan-aware optimization for analytics queries," in *Proceedings of USENIX OSDI*, 2016.

[18] "Measuring internet congestion: A preliminary report," https://ipp.mit.edu/sites/default/files/documents/Congestion-handout-final.pdf,2014.

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

[20] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of USENIX NSDI*, 2012.

[21] "How map and reduce operations are actually carried out," http://wiki.apache.org/hadoop/HadoopMapReduce.

[22] Y. Yu, P. K. Gunda, and M. Isard, "Distributed aggregation for data-parallel computing: interfaces and implementations," in *Proceedings of ACM SIGOPS*, 2009.

[23] W. Deng, F. Liu, H. Jin, and C. Wu, "Smartdpss: cost-minimizing multi-source power supply for datacenters with arbitrary demand," in *Proceedings of IEEE ICDCS*, 2013.

[24] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data centers power reduction: A two time scale approach for delay tolerant workloads," in *Proceedings of IEEE INFOCOM*, 2012.

[25] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995, vol. 1, no. 2.

[26] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[27] M. J. Neely, A. S. Tehrani, and A. G. Dimakis, "Efficient algorithms for renewable energy allocation to delay tolerant consumers," in *Proc. of IEEE SmartGridComm*, 2010.

[28] G. B. Dantzig, *Linear programming and extensions*. Princeton university press, 1998, vol. 48.

[29] Y. Nesterov and A. Nemirovskii, *Interior-point polynomial algorithms in convex programming*. Siam, 1994, vol. 13.

[30] D. Xu, X. Liu, and Z. Niu, "Joint resource provisioning for internet datacenters with diverse and dynamic traffic," *IEEE Transactions on Cloud Computing*, vol. 5, no. 1, pp. 71–84, 2014.

[31] C. Reiss, J. Wilkes, and J. Hellerstein, "Google cluster-usage traces," http://code.google.com/p/googleclusterdata.

[32] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler, "Gupt: privacy preserving data analysis made easy," in *Proc. of ACM SIGMOD*, 2012.

[33] B. Heintz, A. Chandra, and R. K. Sitaraman, "Trading timeliness and accuracy in geo-distributed streaming analytics," in *Proceedings of ACM SoCC*, 2016.

[34] H. Wang, D. Niu, and B. Li, "Turbo: Dynamic and decentralized global analytics via machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1372–1386, 2020.

[35] M. W. Convolbo, J. Chou, C.-H. Hsu, and Y. C. Chung, "Geodis: towards the optimization of data locality-aware job scheduling in geo-distributed data centers," *Springer Computing*, vol. 100, no. 1, pp. 21–46, 2018.

[36] J.-H. Hwang, U. Cetintemel, and S. Zdonik, "Fast and reliable stream processing over wide area networks," in *Proceedings of the IEEE 23rd International Conference on Data Engineering Workshop*, 2007.

[37] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *Proceedings of IEEE ICDE*, 2006.

[38] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing timeliness and cost in geo-distributed streaming analytics," *IEEE Transactions on Cloud Computing*, 2017.

[39] D. Kumar, J. Li, A. Chandra, and R. Sitaraman, "A ttl-based approach for data aggregation in geo-distributed streaming analytics," *Proceedings of ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 2, pp. 1–27, 2019.

[40] F. Liu, Z. Zhou, H. Jin, B. Li, B. Li, and H. Jiang, "On arbitrating the power-performance tradeoff in saas clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2648–2658, 2014.

[41] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with karuna," in *Proc. of ACM SIGCOMM*, 2016.

[42] W. Li, X. Zhou, K. Li, H. Qi, and D. Guo, "Trafficshaper: shaping inter-datacenter traffic to reduce the transmission cost," *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, pp. 1193–1206, 2018.

Xinping Xu is currently a Ph.D. candidate in the School of Computer Science and Technology, Dalian University of Technology, China. His research interests include datacenter networks and cloud computing.

Wenxin Li received the B.E. degree from the School of Computer Science and Technology, Dalian University of Technology, China, in 2012. Currently, he is a Ph.D. candidate in the School of Computer Science and Technology, Dalian University of Technology, China. His research interests include datacenter networks and cloud computing.

Renhai Xu received the B.E. degree from the School of Computer Science and Technology, Dalian University of Technology, China, in 2014. Currently, he is a third-year master student in the School of Computer Science and Technology, Dalian University of Technology, China. His research interests include datacenter networks and cloud computing.

Heng Qi was a Lecture at the School of Computer Science and Technology, Dalian University of Technology, China. He got bachelor's degree from Hunan University in 2004 and master's degree from Dalian University of Technology in 2006. He servered as a software engineer in GlobalLogic-3CIS from 2006 to 2008. Then he got his doctorate degree from Dalian University of Technology in 2012. His research interests include computer network, multimedia computing, and mobile cloud computing. He has published more than 20 technical papers in international journals and conferences, including ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMCCAP) and Pattern Recognition (PR).

Keqiu Li received the bachelors and masters degrees from the Department of Applied Mathematics at the Dalian University of Technology in 1994 and 1997, respectively. He received the Ph.D. degree from the Graduate School of Information Science, Japan Advanced Institute of Science and Technology in 2005. He also has two-year postdoctoral experience in the University of Tokyo, Japan. He is currently a professor in the School of Computer Science and Technology, Dalian University of Technology, China. He has published more than 100 technical papers, such as IEEE TPDS, ACM TOIT, and ACM TOMCCAP. He is an Associate Editor of IEEE TPDS and IEEE TC. He is a senior member of IEEE. His research interests include internet technology, data center networks, cloud computing and wireless networks.

Xiaobo Zhou recieved the B.Sc. in Electronic Information Science and Technology from University of Science and Technology of China (USTC), Hefei, China, in 2007, the M.E. in Computer Application Technology from Graduate University of Chinese Academy of Science (GUCAS), Beijing, China, in 2010, and the Ph.D. degree from School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan, in 2013. He is currently an Associate Professor with the School of Computer Science and Technology, Tianjin University. Prior to that, he was a researcher with Centre for Wireless Communications, University of Oulu, Finland from 2014 to 2015. His research interests include joint source-channel coding, cooperative wireless communications, network information theory, cloud computing and software defined networking.