Optimizing the Cost-Performance Tradeoff for Geo-distributed Data Analytics with Uncertain Demand

Wenxin Li*, Renhai Xu*, Heng Qi*†, Keqiu Li*, Xiaobo Zhou‡, Wenyu Qu‡
*School of Computer Science and Technology, Dalian University of Technology, China.
†Graduate School of Information Science, Nagoya University, Japan.
‡Tianjin Key Laboratory of Advanced Networking, School of Computer Science and Technology, Tianjin University, China.
Heng Qi is the corresponding author: hengqi@dlut.edu.cn.

***Abstract*—In the era of global-scale services, analytical queries are performed on datasets that span multiple data centers (DCs). Due to the scarce and expensive inter-DC bandwidth, various methods have been proposed to reduce either the traffic cost or the completion time for those analytics queries. However, current methods make no attempt to maximize the number of successfully served query requests. Moreover, most of them rely on unrealistic assumptions — such as analytical queries are repeated or known in advance. In this paper, we target at characterizing and optimizing the cost-performance tradeoff for 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. To achieve these objectives, we take advantage of Lyapunov optimization techniques to design 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. Extensive trace-driven simulation results demonstrate that our framework is capable of reducing inter-DC traffic cost, improving system throughput and guaranteeing a maximum delay for each query request.**

I. 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, 4]. These data are born and stored in multiple DCs, and introduce an interesting research topic of geo-distributed data analytics (GDA) [5–7]. Such GDA has huge impact on 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.

While recognizing the significance of GDA, it faces two challenges. *First*, GDA incurs substantial traffic cost, no matter it is 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, the centralized way transfers large volume of input data among multiple DCs, while the distributed way also generates massive cross-DC intermediate data when continuously receiving a large number of query requests. *Second*, modern 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 [8]. While the query demands are rapidly growing, inter-DC network capacity growth has been decelerating [6], making the cross-DC bandwidth become the performance bottleneck for the query requests. This eventually leads to high queuing delay for each query request.

In response to these challenges, we believe that it is crucial to characterize and optimize a *cost-performance* 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 a 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. 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-performance* tradeoff 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 by leveraging efficient data replication and task aggregation strategies for GDA queries [6, 9]; *2)* The second one is to shorten the completion time of GDA query jobs via efficient data/task placement and flow scheduling strategies [5, 7, 10]. However, they make no attempt to maximize the number of successfully served query requests. Moreover, they rely on assumptions that are often unrealistic — such as analytical queries are repeated or known in advance.

In this paper, we make the first attempt to study the *cost-performance* 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

978-1-5386-2704-4/17/$31.00 ©2017 IEEE

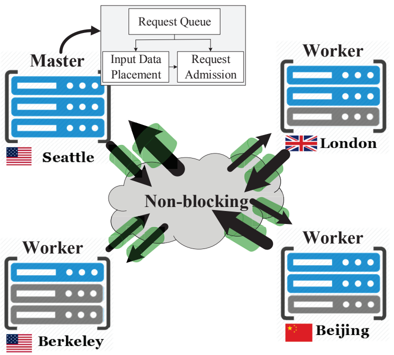


Fig. 1. An illustrative example of a GDA system.

input data placement and query request admission techniques. Specially, a stochastic optimization problem is formulated and solved by an efficient two-timescale online control framework based on Lyapunov optimization techniques. Without a 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. We conduct extensive simulations based on 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 rest of this paper is organized as follows. Section II presents the problem statement and our system model. In Section III, we present our two-timescale online control framework. We present the performance evaluation in Section IV. Section V summarizes the related work. Finally, the conclusions are drawn in Section VI.

II. Problem Statement and System Model

A. Problem Statement

In geo-distributed analytics, multiple DCs are connected to a non-blocking network, via dedicated uplinks and downlinks, as shown in Fig. 1. The bottlenecks are only between DCs and the non-blocking core, which is reasonable because of recent studies and measurements [5, 11]. Data can be generated on any DCs and as such, a dataset could be distributed across many DCs. Each dataset can only be used for one type of 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*.

The problem this paper studied is to design an algorithm to make decisions on both the input data placement and the admission control of query requests in an online manner. 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.

B. Basic 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. 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 [12, 13], 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 a same duration $\hbar$, typically 5 or 15 minutes.

*1) **Online Control decisions**:* 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. *Second*, 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 important control decision is to determine how many requests, $S_i(\tau)$, can be simultaneously served for each query type, in time slot τ. The rest 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.

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

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 amount of unserved query requests at time τ.

*2) **Constraints**:* We have the following constraints.

Ensuring the completion of input data placement: Denote *the amount of input data* to be moved out of (or to be moved in) DC j for i-th query type in t as $f_{i,j}^1(t)$ (or $f_{i,j}^{1'}(t)$).

$$f_{i,j}^1(t)=\max(D_{i,j}(t-T)-D_{i,j}(t),0), \tag{2}$$
$$f_{i,j}^{1'}(t)=\max(D_{i,j}(t)-D_{i,j}(t-T),0). \tag{3}$$

Such data movements must be completed before executing the query requests. Here, we enforce the input data movement to be completed within one fine-grained time slot.

$$\sum_{i=1}^{N}\frac{f_{i,j}^1(t)}{U_j^u}\le\hbar,\forall j,\forall t, \tag{4}$$
$$\sum_{i=1}^{N}\frac{f_{i,j}^{1'}(t)}{U_j^d}\le\hbar,\forall j,\forall t, \tag{5}$$
$$\sum_{j=1}^{M}D_{i,j}(t)=D_i,\forall i,\forall t. \tag{6}$$

Eq. (4) constrains the outward data movement, while Eq. (5) constrains 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). It should be noted that, during these data movements, no query requests can be served, as shown in the following

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

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

Guaranteeing queuing delay of query requests: The queuing delay is closely related to the queue backlog, and hence we bound the length of 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 III-A. So, when the queue backlog is bounded, the maximal delay can also be guaranteed [12].

Link capacity constraint: Both the uplink and downlink bandwidth capacities should be satisfied when transferring the *intermediate data* generated by query requests. Specifically, define $f_{i,j}^2(\tau)$ 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

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

Here, the term $D_{i,j}(t)/D_i$ represents the fraction of reduce tasks assigned to DC j, which is proportional to the amount of input data that placed on it. It should be noted that we simply use such proportional task placing strategy to derive the distribution of the intermediate data. One can further reduce the cost with better task placements [5, 6], but it is beyond the scope of this paper.

Similarly, we also define the amount of intermediate data to be downloaded as $f_{i,j}^{2'}(\tau)$, which is calculated as follows

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

We now can formulate the link capacity constraint with the following inequalities satisfied:

$$\sum_{i=1}^{N} \frac{f_{i,j}^2(\tau)}{\hbar} \le U_j^u, \forall j, \forall \tau, \tag{13}$$

$$\sum_{i=1}^{N} \frac{f_{i,j}^{2'}(\tau)}{\hbar} \le U_j^d, \forall j, \forall \tau. \tag{14}$$

3) ***Characterizing the Cost-Performance Tradeoff****:* We have two objectives, as shown in the following.

Inter-DC traffic cost: For modern GDA system, it is crucial to minimize the inter-DC traffic cost incurred by a large amount of query requests. The inter-DC traffic cost contains two parts: the traffic cost of input data movement and the traffic cost of intermediate data transmission. It should be noted that we do not enforce any complicated traffic pricing model, e.g., 95th percentile pricing. The traffic cost is simply 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

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

Define $C(\tau) \triangleq \sum_{i=1}^{N}\sum_{j=1}^{M}\left(\frac{1}{T}f_{i,j}^1(t) + f_{i,j}^2(\tau)\right)$, 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.

System throughput: For large-scale GDA system, another important performance metric 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 τ, the system throughput is simply defined as the summation of served requests across all query types

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

With the above objectives, the cost-performance 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 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 the 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 [14], which, however, requires significant knowledge of the query request. Hence, we are motivated to take advantage of the Lyapunov framework [15] to design online control algorithms, without requiring a prior knowledge of the query requests.

III. A Two-timescale Online Control Framework

A. Transforming into Lyapunov Optimization

The key of 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, we construct a group of delay-aware virtual queues $Y_i(t), \forall i$, based on the technique of ϵ-persistent queue [16]. The queuing dynamics is defined as:

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

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 is query requests in $Q_i(t)$ that has not been serviced. The following Lemma shows that the deferred query requests should be served within a worst case delay l_{max} under any feasible algorithm. Due to the page limit, we omit the proof process.

***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 requests can be served within a maximal queuing delay $l_{max} = \lceil (Q_{max} + Y_{max})/\epsilon \rceil$.*

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:

$$L(\mathbf{\Theta}(t)) = \frac{1}{2}\left(\sum_{i=1}^{N} Y_i^2(t) + \sum_{i=1}^{N} Q_i^2(t)\right) \tag{19}$$

This Lyapunov function quantitatively reflects the congestion of all queues [15], 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

$$\Delta_T(\mathbf{\Theta}(t)) = \mathbb{E}\{L(\mathbf{\Theta}(t+T)) - L(\mathbf{\Theta}(t))|\mathbf{\Theta}(t)\} \tag{20}$$

Based on the Lyapunov framework [15], 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

$$\Delta_T(\mathbf{\Theta}(t)) + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\left(C(\tau) - \lambda S(\tau)\right)|\mathbf{\Theta}(t)\} \tag{21}$$

where $V(\geq 0)$ is a control parameter representing how much we emphasize the cost-performance tradeoff (Problem **P1**) compared to the system stability.

B. Two-timescale Online Control Algorithm

To design an online control algorithm, an intuitive approach is to derive the upper bound of the *drift-plus-penalty* term, and minimize such upper bound 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 τ∈$[t, t$+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. The following theorem gives an upper bound for such *drift-plus-penalty* term. We omit the proof process in this paper due to the page limit.

***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*

$$\begin{aligned}
&\Delta_T(t) + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\left(C(\tau) - \lambda S(\tau)\right)|\mathbf{\Theta}(t)\} \\
&\leq HT + V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\left(C(\tau) - \lambda S(\tau)\right)|\mathbf{\Theta}(t)\} \\
&+ \mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Y_i(t)\left(\epsilon - S_i(\tau)\right)|\mathbf{\Theta}(t)\} \\
&+ \mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Q_i(t)\left(A_i(\tau) - S_i(\tau)\right)|\mathbf{\Theta}(t)\}
\end{aligned} \tag{24}$$

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

By substituting the definitions of $C(\tau)$ and $S(\tau)$ into the right hand side of Eq. (24), we get the following relaxed problem **P2**:

$$\begin{aligned}
&\min V\mathbb{E}\{\sum_{i=1}^{N}\sum_{j=1}^{M}\max\{D_{i,j}(t-T) - D_{i,j}(t), 0\}|\mathbf{\Theta}(t)\} \\
&+ V\mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N}\sum_{j=1}^{M} D_{i,j}(t)\alpha_i(1 - \frac{D_{i,j}(t)}{D_i})S_i(\tau)|\mathbf{\Theta}(t)\} \\
&- \mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N}(Y_i(t) + V\lambda)S_i(\tau)|\mathbf{\Theta}(t)\} \\
&+ \mathbb{E}\{\sum_{\tau=t}^{t+T-1}\sum_{i=1}^{N} Q_i(t)\left(A_i(\tau) - S_i(\tau)\right)|\mathbf{\Theta}(t)\}
\end{aligned} \tag{25}$$

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

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:
       $\min_{D_{i,j}(t)} V\mathbb{E}\{\sum_{i=1}^{N}\sum_{j=1}^{M}\max\{D_{i,j}(t-T) - D_{i,j}(t), 0\}|\mathbf{\Theta}(t)\}$
       $+ VT\mathbb{E}\{\sum_{i=1}^{N}\sum_{j=1}^{M} D_{i,j}(t)\alpha_i(1 - \frac{D_{i,j}(t)}{D_i})Q_i(t)|\mathbf{\Theta}(t)\}$
       Subject to: Eqs. (4), (5), (6).    (22)
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:
       $\min_{S_i(\tau)} \mathbb{E}\{\sum_{i=1}^{N} Q_i(t)\left(A_i(\tau) - S_i(\tau)\right)|\mathbf{\Theta}(t)\}$
       $- \mathbb{E}\{\sum_{i=1}^{N}(Y_i(t) + V\lambda)S_i(\tau)|\mathbf{\Theta}(t)\}$    (23)
       Subject to: Eqs. (1), (7), (8), (9), (10), (13), (14)
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.

IV. Performance Evaluation

A. Simulation setup

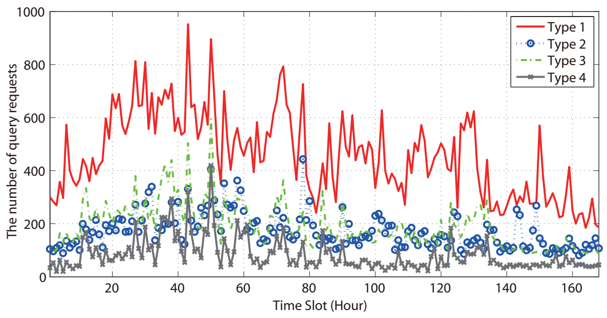


Fig. 2. 7-day real world traces from [17].

Datasets: To simulate the arrival patterns of query requests, we use Google cluster traces [17], 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. 2 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 [5]. 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 are 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. In order to study the impact of weight factor λ on the performance of our algorithm, we fix the values of other parameters to be constants, e.g. $T = 10$, $V = 10$, $\epsilon = 1$.

Compared algorithms: We compare our *2TGDA* with two 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*". Another algorithm is an online algorithm that always schedules query requests immediately regardless of the inter-DC traffic cost, which is referred as "*Impatient*".

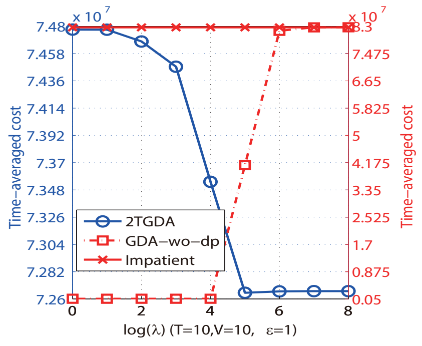
B. Simulation results

We mainly evaluate three performance metrics: time-averaged cost, time-averaged throughput and queuing delay. The simulation results are summarized in Fig. 3. We can clearly observe that no matter how the weight factor λ changes, the performance metrics achieved by the *Impatient* algorithm will never change. This is because that *Impatient* algorithm immediately schedules the query requests, as long as meeting the link capacity. For the GDA-wo-dp algorithm, the achieved time-averaged cost increases as the weight factor λ increases, as shown in Fig. 3(a). The reason is that *GDA-wo-dp* also studies a cost-performance problem, where λ emphasizes the importance of the system throughput maximization. This eventually enforces *GDA-wo-dp* to achieve a higher cost with a larger value of λ. Actually, this is also why it achieves an increasing throughput as λ increases, as shown in Fig. 3(b). We can further observe that *GDA-wo-dp* guarantees a decreasing value of maximum queuing delay l_{max} as λ increases, as shown in Fig. 3(c). This is mainly because that a larger value of λ leads to a larger throughput, making fewer query requests to be blocked. An interesting observation is that the time-averaged cost achieved by our *2TGDA* algorithm decreases as the increasing of λ which is actually 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 saving if there are more concurrent query requests. So, that is why *2TGDA* achieves an increasing time-averaged throughput with the increasing of the weight factor λ. Finally, with the increasing of λ, the maximum queuing delay l_{max} achieved by our *2TGDA* algorithm can be controlled slowly increase at the beginning, fast increase in the middle, and finally close to a stable value.

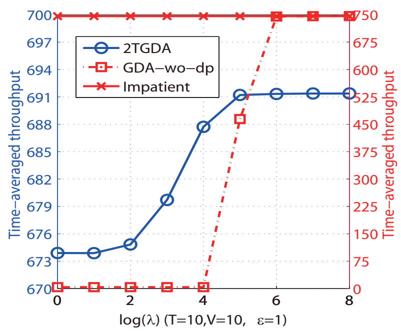
The above results show that our *2TGDA* algorithm can simultaneously reduce the traffic cost, improve the system throughput, and provide a maximal delay guarantee for each query request, by choosing a sufficient large value of the weight factor λ (e.g., $\lambda = 10^5$).

V. Related Work

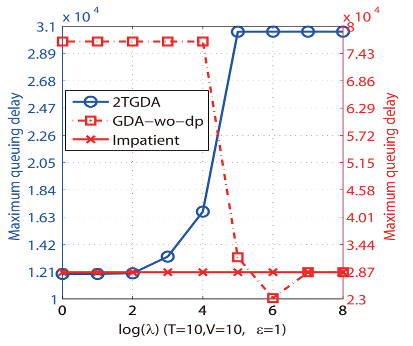
Existing work in geo-distributed data analytics can be divided into two categories based on their objectives. Regarding the inter-DC traffic cost, Vulimiri et al. solve an integer linear program to optimize the query execution plan and aggressively cache the results of prior queries for the subsequent queries [6]. Pixida is a scheduler that takes advantage of the graph partition method to minimize the inter-DC data transfers [9]. The above solutions do not consider any performance issues related to the geo-distributed data analytics queries, and also ignore the relationship between input data and the intermediate data generated by many queries. Regarding the job completion



(a) Time-averaged cost vs. λ



(b) Time-averaged throughput vs. λ



(c) Maximum queuing delay vs. λ

Fig. 3. The performance of (a) time-averaged cost, (b) time-averaged throughput, (c) maximum queuing delay, for different methods under different values of weight factor λ.

time, Iridium optimizes the placement of both the input data and reduce task [5]. Hung et al. propose SWAG, which leverages a greedy job scheduling algorithm to optimize the average job completion time [10]. CLARIENT is a recently proposed solution for reducing the query completion time, which use heuristics to jointly selects the placements and schedules of tasks [7]. However, all of they make no attempt to maximize the number of successfully served query requests. Moreover, they rely on assumptions that are often unrealistic — such as analytical queries are repeated or known in advance (e.g., [5, 7]), or each DC serves one task per second [10].

VI. Conclusions

This paper studies a cost-performance problem for geo-distributed data analytics, with the aim of minimizing the inter-DC traffic cost as well as maximizing the number of query requests that can be successfully served with guaranteed queuing delay. To this end, we take advantage of Lyapunov optimization to design a two-timescale online control algorithm. Without requiring a prior knowledge of subsequent query requests, this algorithm makes online decisions on both the input data placement and the query request admission. We use trace-driven simulation to verify that our algorithm is effective in arbitrating the cost-performance tradeoff and guaranteeing a maximal queuing delay for each query request.

Acknowledgment

This work is supported by the National Key Research and Development Program of China No. 2016YFB1000205, the State Key Program of National Natural Science of China (Grant No. 61432002), the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61225010), NSFC Grant Nos. 61272417, 61300189, 61370199 and 61672379; Specialized Research Fund for the Doctoral Program of Higher Education (Grant No. 20130041110019), and the Fundamental Research Funds for the Central Universities (Grant. DUT15QY20); the Dalian High-level Talent Innovation Program (No. 2015R049), and the JSPS KAKENHI under Grant 16F16349.

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 *Proc. 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 *Proc. of USENIX NSDI*, 2014.

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

[6] 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 *Proc. of USENIX NSDI*, 2015.

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

[8] 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 *Proc. of ACM SIGMOD*, 2013.

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

[10] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proc. of ACM SoCC*, 2015.

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

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

[13] 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 *Proc. of IEEE INFOCOM*, 2012.

[14] 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.

[15] 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.

[16] 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.

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