Optimizing the Performance-Cost Tradeoff in Cross-Edge Analytics

Lin Jia*, Zhi Zhou† and Hai Jin*
*Service Computing Technology and System Lab, Cluster and Grid Computing Lab, Big Data Technology and System Lab
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
†School of Data and Computer Science, Sun Yat-sen University, Guangzhou, 510006, China
Email: {ljia, hjin}@hust.edu.cn, zhouzhi9@mail.sysu.edu.cn

***Abstract*—As an emerging computing paradigm, edge computing has gathered much attention recently. With edge computing, application datasets as examplified by performance logs and activity records naturally span across multiple edge sites. Analyzing such cross-edge datasets is however, by no means trivial, since the status quo approach which aggregates the raw data to a centralized cloud datacenter incurrs high performance and cost overheads. To address this challenge, in this paper, we tackle the problem of speeding up cross-edge analytics with low traffic cost, through joint optimization of task and input data placement. The resulted performance-cost tradeoff problem is difficult due its non-convexity and the uncertainty of query characteristics. To address these challenges, we combine convex relaxation with a two-stage optimization. Specifically, a prediction of the query characteristics is used to determine the data movement when the data is generated, and then the actual value is used to decide the task placement when the query arrives. Evaluations using a production trace from a Facebook cluster highlight that, the two-stage joint optimization approach can reduce the total cost by up to 83% compared to the status quo approach.**

I. Introduction

Edge computing (EC) [1], [2], with its promise to fulfill the urgent need for richer applications and better experience of resource-hungry mobile devices, is emerging as a new computing paradigm and ascending to the spotlight. In edge computing, cloud computing capabilities and service environments are pushed from the internet core to the edge of the cellular network. By running applications and processing tasks in closer proximity to the end devices and users to greatly reduce the end-to-end latency, edge computing is boosting many emerging applications including video/audio surveillance, remote e-health care, *Internet of Vehicles* (IoV), augmented reality, etc [3], [4].

With multiple edge computing sites spanning across the network edge, application data such as activity records of a social networking services, and performance logs of a web search engine — is naturally initiated at each geo-distributed edge node [5]. How to analyze such geo-distributed datasets is undergoing a revolution. Traditionally, the "centralized" approach would aggregate the geo-distributed datasets to a centralized, well-provisioned datacenter before analyzing them locally. Given the unprecedented and continuous growth in data volumes, this centralized aggregation would not only incur significant amount of *wide-area network* (WAN) traffic, but also increase waiting time for analytic jobs.

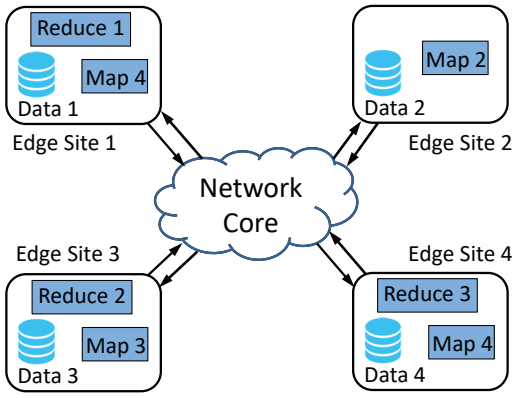


Fig. 1. An example of cross-edge Map-Reduce query

In response to the above challenges, we propose *cross-edge analytics* for edge computing in this paper. With cross-edge analytics, the computation tasks of both input stage (e.g., map) and output stage (e.g., reduce) are pushed to the edge sites where data was born, rather than aggregating the raw data to a centralized datacenter. Fig. 1 shows an example of cross-edge Map-Reduce query. In this example, both map and reduce tasks are moved to where the data is stored. As a result, both query response time and WAN traffic can be significantly reduced. *However, the benefit of cross-edge analytics can not be fully materialized if we leave the current analytics frameworks such as Map-Reduce and Spark unmodified* since these frameworks are designed to operate in intra-cluster environments, in which the bandwidths of different nodes are relatively identical. While in cross-edge setting, the up- and downlink bandwidths at each edge site are usually the scarcest and most volatile resource, thus the duration of the intermediate data transfer across the sites may exhibit great variability. However, for frameworks such as Map-Reduce [6] and Spark [7], only after all the up- and download transfers for the intermediate stage are finished, the next stage can be started. Therefore, it is critical to reduce the duration of the slowest data transfer if we want to *further* reduce the response time of the query, this in turn requires us to orchestrate the intermediate data transfers across those edge sites, by taking advantage of the latter's bandwidth heterogeneity.

In this paper, we study *how to speed up geo-distributed analytics by making use of the bandwidth heterogeneity across*

the edge sites. The duration of the intermediate stage which dominates the job response time, is *jointly* determined by the amount of input data and reduce task placed at each edge site. Then an intuitive approach towards the above goal is to optimize the task placement and input data placement (via moving input data among edge sites) *simultaneously*. In particular, by coordinating the input data placement and task placement, we carefully adapt the amount of data transferred at the *source* and *destination* of intermediate data flows, respectively, to balance the duration of each transfer as much as possible. As a result, the duration of the slowest transfer and thus the intermediate stage can substantially reduced. Unfortunately, the benefit of input data placement can not be earned without pay, as the latter would lead to increased WAN traffic cost. Consequently, when performing joint optimization on input data and task placement, a fundamental problem is to *reduce the query response time at a low traffic cost*.

To solve the above problem, in this paper we develop a practical performance-cost trade-off model to study the problem, and a two-stage algorithm is proposed to optimize the performance-cost trade-off. Minimizing query response is in conflict with minimizing traffic cost, since the former requires more input data movement. Thus we need to strike a balance between the performance in terms of response time and the traffic cost. Towards this goal, we firstly construct a unified cost objective to couple the two potentially conflicting objectives, response time and traffic cost, in an economic way. Consequently, the performance-cost trade-off problem can be formulated as a cost minimization problem for joint input data and task placement. However, solving the cost minimization problem is rather challenging, due to: (1) the problem is non-convex which rules out the direct application of existing solvers for convex problems. (2) When making the decision on data movement before the query arrives, we do not have the exact value of the query selectivity (*i.e.*, the ratio of the amount of intermediate data produced to that of the input data), even if predictions could be utilized, then the open question is how to best minimize the effect of prediction error on the cost objective?

To address the above challenges, we propose an efficient solution by combining the advantages of convex relaxation and a two-stage optimization. Specifically, we firstly convexify the non-convex problem by omitting the constraint that the amount of intermediate data transferred from one site to others should be proportional to be fraction of reduce task placed at the latter. Note that the resulted data movement would make the most use of variability on bandwidth availability and traffic cost *e.g.*, moving data out of the bottleneck sites with low bandwidth and high cost. The data movement is still beneficial (though may not be optimal) if we re-consider the aforementioned proportionality constraint. We further handle the uncertainty on query selectivity by employing a two-stage optimization approach, at the first stage when input data is generated, we determine the input data movement by solving the convex relaxation problem with a predicted selectivity. Then at the second stage when query arrives, we solve a separate linear task placement problem to correct the query response time as well as the task placement, based on the exact value of the query selectivity.

II. Related Work

Big-Data Analytics in Cloud Computing. As a result of the unprecedented increase in data volume, the problem of big-data analytics in cloud datacenters is not a new topic and has been extensively studied. Most of existing works focus on optimizing single objective on either query response time or WAN bandwidth usage. For example, Pixida [8], a system aims at minimizing the time of data movement with the bandwidth-constrained links. JetStream [9] is proposed to reduce the bandwidth requirement at bottleneck links, by pre-processing the data locally before aggregating to a central datacenter. Under the similar geo-distributed analytics context, recent works [10] optimize the WAN traffic, while [11] reduces average completion time of jobs by coordinating the task execution at each datacenter.

Big-Data Analytics in Edge Computing. With the proliferation of edge computing, big-data analytics in edge computing has begun to recieve great attention recently. To satisfy the strict requirements of real-time and large-scale video analytics, a hybrid and cost-efficient architecture based on both cloud and edge is proposed [4]. To provide low-latency video analytics at places closer to the users, an edge computing-based system is built to offload computation tasks between clients and edge nodes and collaborate nearby edge nodes [12]. For the long-studied problem of traffic estimation, a fine-grained and edge-based traffic volume estimation scheme which uses in-vehicle dashboard mounted cameras is proposed [13]. Our work is different from and complementary to these works in at least two important aspects: First, we do not assume the query selectivity as a priori. Second, we formulate the underlying problem to provide insight for both in-depth understanding and efficient solution.

III. System Model for Cross-edge Analytics

In this part, we present the architecture, system model and optimization for cross-edge analytics. The notations used in this section are summarized in Table I.

A. Overview of the Cross-Edge Infrastructure

We consider the cross-edge analytics framework running on a set of S geographically dispersed edge sites, denoted as $\mathcal{S} = \{1, 2, \cdots, S\}$. In line with the most recent systematic work on big data analytics [14], we assume that these sites are inter-connected by a wide-area network. Compared with the network core which has abundant bandwidth, the available bandwidths of the links between the network core and the edge sites are usually limited and are more likely to be the bottlenecks, as validated by recent measurements [15]. Moreover, the uplink and downlink bandwidths at those edge sites cloud also exhibit significant heterogeneity. In this paper, we use U_i and D_i to denote the amount of available bandwidth of the up- and downlink at edge site $i \in \mathcal{S}$, respectively.

TABLE I
KEY NOTATIONS

Notation	Definition
$\mathcal{S}$	Set of the edge sites, $\lvert\mathcal{S}\rvert = S$
U_i	Uplink bandwidth at edge site i
D_i	Downlink bandwidth at edge site i
r_i	Fraction of reduce task placed at edge site i
M_i	Amount of input data originated at edge site i
c_i^U	Duration of the upload transfer at edge site i
c_i^D	Duration of the download transfer at edge site i
α	Selectivity of the query
z	Duration of the shuffle phase
d_i^U	Duration of the input data upload at edge site i
d_i^U	Duration of the input data download at edge site i
d_i^U	Duration of the input data download at edge site i
P_i^U	Traffic price for data upload at edge site i
P_i^D	Traffic price for data download at edge site i
L	Lag between the generation of input data and the arrival of its query

At each edge site, we also assume that the compute and storage capacity are relatively abundant to process and store the data. Note that this assumption is quite reasonable in practice, as compared to the WAN bandwidth capacity whose growth has been decelerating [10], compute and storage capacity can be readily expanded at relatively lower cost.

B. Model of Corss-Edge Analytics

When services are hosted on top of geo-distributed computing sites, large quantities of data such as user records and system logs are continuously generated at each edge site. For a distributed dataset (*e.g.*, user activity records) spanning multiple edge sites, we use M_i to denote the amount of data partition originated at edge site $i \in \mathcal{S}$. When a user query arrives, a logically centralized global manager firstly converts the user query into a DAG (*directed acyclic graph*) of stages, each of which consists of many parallel computing tasks. *In this paper, we consider there are only two computing stages, i.e., map and reduce* [6]. Though simple, it preserves the fundamental communication pattern—shuffle—between the two stages.

Input tasks of a query (*i.e.*, map tasks in this work) can execute locally on edge sites that contain their input data, and their outputs, *i.e.*, the intermediate data are then written to each local edge site. For a specific query, the amount of intermediate data at each edge site $i \in \mathcal{S}$ can be denoted as αM_i, where α is called the selectivity of the query and denotes the ratio of the amount of intermediate data to that of input data. As a result of in-memory caching of data and data locality, the input stage can be finished extremely quick. Then, in the shuffle phase, intermediate data at each edge site is disseminated to the reduce tasks placed across multiple edge sites. Finally, the reduce tasks are executed on the shuffled intermediate data to produce the final outputs of the user query, and return them to the global manager.

Note that the shuffle phase is data-intensive and the data transfers from map tasks to reduce tasks may necessarily incur all-to-all communication across the edge sites. Specifically, given the placement profile of the reduce tasks, r_i, *i.e.*, the fraction of reduce tasks placed at each site $i \in \mathcal{S}$, which are summed to be 1 ($\sum_{i\in\mathcal{S}} r_i = 1$), then the intermediate data transferred out of edge site $i \in \mathcal{S}$ can be denoted as $(1 - r_i)\alpha M_i$. The intermediate data transferred into edge site $i \in \mathcal{S}$ can be denoted as $r_i\alpha(M - M_i)$, where M is a constant and denotes the total amount of input data of the query, *i.e.*, $M = \sum_{i\in\mathcal{S}} M_i$. We use c_i^U and c_i^D to denote the duration of the upload and download of intermediate data at edge site i, respectively. Then, for the uplink at each edge site i, we have:

$$U_i c_i^U = (1 - r_i)\alpha M_i \tag{1}$$

Similarly, for the downlink at each site i, we have:

$$D_i c_i^D = r_i\alpha(M - M_i) \tag{2}$$

The duration of the intermediate communications, denoted as z, depends on the duration of the slowest up- or download transfer across the edge sites, *i.e.*,

$$z = \max_{i\in\mathcal{S}} \max\{c_i^U, c_i^D\} \tag{3}$$

As discussed in Sec. III-A, the bandwidth U_i and D_i typically exhibit great heterogeneity across multiple edge sites, as a result, the duration of those upload and download transfers may also show great variability.

C. Optimizing Cross-Edge Analytics

For the cross-edge analytics framework, our goal is to minimize both the query response time and the WAN traffic, as the former is a key measure of service performance/quality, while the latter is a fundamental implication of operational cost ($/bytes).

Given the assumption of abundant CPU and IO resources, WAN is the only bottleneck and the response time of a query is dominated by the duration of the data-intensive shuffle phase. Thus, we focus on minimizing the duration of data-intensive shuffle phase, *i.e.*, z defined in Sec. III-B to reduce the query response time. Unfortunately, minimizing the duration of the shuffle phase is non-trivial, as it is bounded by the slowest up- or download transfer across the edge sites, which means that reduce the duration of some transfers may not necessarily reduce the duration of the shuffle phase.

An intuitive yet promising approach to minimizing the duration of the shuffle phase, is to balance the duration of those upload and download transfers via careful task placement (*i.e.*, r_i defined in Sec. III-B). We illustrate this intuition with a 2-edge example, the setup is shown in Table II. In the example, as r_1 changes, the duration of the transfer on each link is plotted in Fig. 2. We can see that, by tuning r_1 (and thus r_2), we can change the duration of each transfer, specifically, when $r_1 = 0.57$, the duration of the slowest transfer is minimized, with the value of 8.57s. While when r_1 deviates from the optima, the durations of the transfers become more unbalanced and lead to a larger duration of shuffle phase.

TABLE II
SETUP OF THE 2-EDGE EXAMPLE WITH SELECTIVITY $\alpha = 1$

	Input Data (MB)	Uplink (MB/s)	Downlink (MB/s)
Edge 1	200	10	25
Edge 2	300	20	20

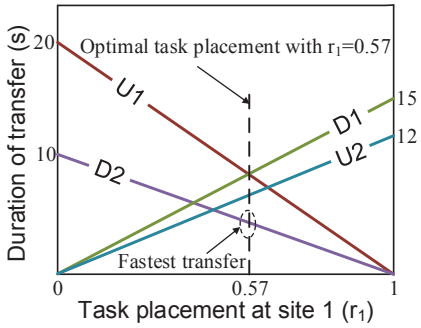


Fig. 2. Without input data movement, the duration of each transfer under various r_1. Here U1 denotes the upload transfer at edge site 1, similarly for U2, D1, and D2.

Though careful task placement can reduce query response time via balancing the duration of each transfer at different WAN links, the optimal task placement is still constrained by the distribution of the intermediate data. This can be seen from the Eq. (1) and (2), where the amount of data transferred at each WAN link jointly depends on the task placement r_i and the distribution of the input data M_i. Thus, data placement (*i.e.*, changing the distribution of input data at each edge site) is complementary to task placement towards reducing query response time. Again, in Fig. 2, we can see that at the optima $r_1 = 0.57$, the download transfer at site 2 has the smallest duration while the download transfer at edge site 1 has the largest duration, then a natural question is, can we move some input data from edge site 2 to edge site 1 to accelerate the shuffle phase? To answer this question, we try different amount of input data to be moved from edge site 2 to edge site 1, and examine the corresponding duration of the shuffle phase, Fig. 3 illustrates the trade-off between the duration of the shuffle phase and the amount of input data movement from edge site 2 to site 1. Interestingly, we can see that, the duration of shuffle phase diminishes as the amount of input data moved, and it reduces to 0 if all the input data at edge site 2 is moved to edge site 1. Furthermore, the figure also shows that the duration of shuffle phase decreases faster as the amount of data moved increases, demonstrating the increasing marginal benefit (the reduction of the duration brought by each additional unit of data moved) of input data movement.

Unfortunately, even with the benefits of reducing the query response time, data movement however increases the amount of WAN traffic used for data transfer. As a result, a fundamental challenge in performing joint data and task placement, is how to optimize the trade-off between the above the two potentially conflicting objectives in a cost-effective manner. To this end, we firstly construct a unified cost objective to couple these conflicting sides in an economic way.

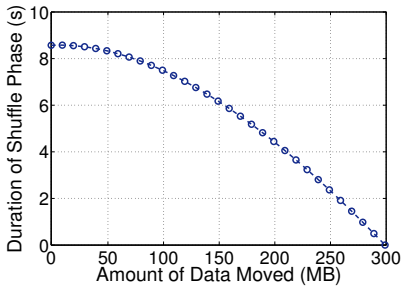


Fig. 3. The duration of the shuffle phase diminishes as the amount of input moved from edge site 2 to edge site 1 increases

D. Arbitrating the Cost-Performance Tradeoff

We optimize the cost-performance trade-off by coupling the response time and traffic usage into a unified cost objective. Specifically, for interactive web-applications such as web search and social networking, even a small increase in latency can significantly impact the revenue of service providers, as demonstrated by the measurements conducted by internet giants [16], [17]. For Google, an additional 400 ms latency in search responses reduces search volume by 0.74%. For Amazon, a 100 ms latency increase implies a 1% sales loss. For Microsoft Bing, a 500 ms latency increase would lead to 1.2% revenue loss. Here we use *delay cost* to refer to the economic loss due to the increased response time [18], and adopt a linear loss function θz to capture this delay cost, the parameter θ is the price that converts response time to a monetary term.

On the other hand, when moving input data and transferring intermediate data across the edge sites, network traffic cost is involved. In reality, there are two major kinds of charging scheme for network traffic. In the first scheme, the traffic cost is linear with the traffic volume. While the second is the 95-percentile charging scheme which is mainly adopted in ISP network. In this paper, we consider the former, *i.e.*, the linear charging scheme, since optimizing the linear cost can also efficiently reduce the 95-percentile cost [19]. Given the amount of input data $U_i d_i^U$ ($D_i d_i^D$) moved out of (into, Resp.) edge site i, where d_i^U (d_i^D) is the duration of input data upload (download, Resp.), and the price P_i^U (P_i^D) for uploading (downloading, Resp.) one unit of data at edge site i. The traffic cost for moving input data and transferring intermediate data can be denoted as: $\sum_{i\in\mathcal{S}}\{P_i^U U_i(c_i^U+d_i^U)+P_i^D D_i(c_i^D+d_i^D)\}$.

Given the revenue loss brought by increased response time and the cost incurred by WAN traffic, the following total cost

$$\theta z + \sum_{i\in\mathcal{S}}\{P_i^U U_i(c_i^U + d_i^U) + P_i^D D_i(c_i^D + d_i^D)\} \qquad (4)$$

arbitrates the aforementioned performance-cost trade-off is expected to be minimized, via joint data and task placement.

Specifically, after the data movement, the amount of input at each edge site i is $M_i+D_i d_i^D-U_i d_i^U$, thus the amount of data to upload from site i is $(1-r_i)\alpha(M_i+D_i d_i^D-U_i d_i^U)$, and the amount of data download to edge site i is $r_i\alpha\{\sum_{i'\in\mathcal{S}}(M_{i'} + D_{i'} d_{i'}^D - U_{i'} d_{i'}^U) - (M_i + D_i d_i^D - U_i d_i^U)\}$. Since the total

amount of data upload should be equal to that of data download across multiple edge sites, we have $\sum_{i' \in \mathcal{S}} U_{i'} d_{i'}^U = \sum_{i \in \mathcal{S}} D_{i'} d_{i'}^D$, together with the equation $\sum_{i' \in \mathcal{S}} M_{i'} = M$ presented in Sec. III-B, the amount of data download at edge site i can be rewritten as $r_i \alpha (M - (M_i + D_i d_i^D - U_i d_i^U))$. In addition, the data movement should be finished within the query *lag* L, *i.e.*, the time interval between the data generation and query arrival.

We are now in a position to formulate the performance-cost trade-off for cross-edge analytics as an optimization problem that minimizes the total cost consisting of revenue loss due to increased response time, and operational cost incurred by WAN traffic.

$$\begin{aligned}
\min \quad & z + \sum_{i \in \mathcal{S}} \{P_i^U U_i (c_i^U + d_i^U) + P_i^D D_i (c_i^D + d_i^D)\} \\
\text{s.t.} \quad & \sum_{i \in \mathcal{S}} r_i = 1 && \text{(5a)} \\
& U_i c_i^U = (1 - r_i) \alpha (M_i + D_i d_i^D - U_i d_i^U) && \text{(5b)} \\
& D_i c_i^D = r_i \alpha (M - (M_i + D_i d_i^D - U_i d_i^U)) && \text{(5c)} \\
& \sum_{i \in \mathcal{S}} U_i d_i^U = \sum_{i \in \mathcal{S}} D_i d_i^D && \text{(5d)} \\
& 0 \leq d_i^U \leq M_i / U_i && \text{(5e)} \\
& 0 \leq d_i^U \leq L && \text{(5f)} \\
& 0 \leq d_i^D \leq L && \text{(5g)} \\
& 0 \leq c_i^U \leq z && \text{(5h)} \\
& 0 \leq c_i^D \leq z && \text{(5i)} \\
& 0 \leq r_i && \text{(5j)}
\end{aligned}$$

For ease of presentation, we assume that one unit of increased response time leads to one unit of revenue loss, then the price θ is omitted in the objective. (5a) is the constraint that task placement r_i is summed to be 1 across the sites. (5b) and (5g) compute the amount of intermediate data uploaded and downloaded at each edge site i after the data movement, respectively. (5d) implies that the total amount of data upload should be equal to that of data download across those edge sites. (5e) implies that the amount of input data moved out of each edge site can not exceed the original amount M_i. (5f) and (5g) enforce that the input data movement should be finished within the query lag time. (5h) and (5i) implies that the duration of shuffle phase is bounded by the slowest up- or download transfer across the sites, which is equivalent to but of easier presentation than Eq. (3) in Sec. III-B.

Unfortunately, solving the above problem (5) *for even single query* is highly challenging, due to: (1) the above problem is non-convex, as the multiplying of decision variables (*e.g.*, $r_i d_i^D$) makes the constraints (5b) and (5g) non-convex. (2) The above problem involves uncertainly and should be solved as soon as the dataset has been generated, however, the selectivity of the query, α remains unknown until the query arrives. This means that, when solving the above problem, we do not have the exact value of α.

IV. Algorithm for Uncertain Query

In response to the challenges of the performance-cost trade-off problem, we firstly carefully relax the non-convex problem (5) to make it convex and thus tractable. Then, to address the uncertainty of the query selectivity, we take a two-stage optimization approach to decouple data placement from task placement. Specifically, at the first stage (*i.e.*, when data is generated), we solve the convex relaxation problem to obtain a sub-optimal solution for data movement, based on a predicted query selectivity. Then at the second stage when query arrives, we solve the task placement problem with the exact selectivity which is then available.

A. Convex Relaxation for the Performance-Cost Trade-off Problem

Recall that the performance-cost trade-off problem is non-convex since the common term $r_i \alpha (M_i + D_i d_i^D - U_i d_i^U)$ in constraints (5b) and (5c) contains products of decision variables. Interestingly, we can see that this term represents the amount of intermediate data reserved locally at site i. To eliminate the decision variable r_i and thus the products of decision variables, we introduce the new notation t_{ij} to denote the amount of intermediate data transferred from site i to site j, then there is a direct connection between t_{ij} and the original variables d_i^U, d_i^D, and r_j:

$$t_{ij} = \alpha (M_i + D_i d_i^D - U_i d_i^U) r_j \tag{6}$$

Putting it together with the constraint (5a), we obtain:

$$\sum_{j \in \mathcal{S}} t_{ij} = \alpha (M_i + D_i d_i^D - U_i d_i^U) \tag{7}$$

Eq. (7) is intuitive, as it represents that the intermediate data generated at each site is distributed among all the sites. Furthermore, the constraints (5b) and (5c) can now be re-written as:

$$U_i c_i^U = \sum_{j \in \mathcal{S}} t_{ij} - t_{ii} \tag{8}$$

$$D_i c_i^D = \sum_{j \in \mathcal{S}} t_{ji} - t_{ii} \tag{9}$$

While it is critical to note that the transformation (7)-(9) is not equivalent to the original constraints (5a)-(5c), since the proportionality between the amount of intermediate distributed to each site and the fraction of task placed at each site is not enforced. Unfortunately, maintaining this proportionality constraint, *i.e.*,

$$\frac{t_{1i}}{\sum_{j \in \mathcal{S}} t_{1j}} = \frac{t_{2i}}{\sum_{j \in \mathcal{S}} t_{2j}} = \cdots = \frac{t_{Mi}}{\sum_{j \in \mathcal{S}} t_{Mj}} \quad (= r_i) \tag{10}$$

is challenging, since this proportionality constraint is non-convex (due to the products of t_{ij}'s).

For tractability, we ignore this constraint (10). Naively removing this constraint will result in an extreme case where intermediate data produced at each site is entirely reserved locally, as this leads to zero traffic cost and zero response time. To exclude this extreme case and to fully realize the benefit of

input data movement, we enforce that $t_{ii} = 0, \forall i \in \mathcal{S}$, meaning that the intermediate data produced at each site is completely moved out to other sites. We now use the transformation (7)-(9) to replace the original constraints (5a)-(5c), yielding the following convex relaxation problem:

$$\begin{aligned}
\min \quad & z + \sum_{i\in\mathcal{S}} \{P_i^U U_i (c_i^U + d_i^U) + P_i^D D_i (c_i^D + d_i^D)\}, && (11)\\
\text{s.t.} \quad & \sum_{j\in\mathcal{S}} t_{ij} = \alpha(M_i + D_i d_i^D - U_i d_i^U)\\
& U_i c_i^U = \sum_{j\in\mathcal{S}} t_{ij} - t_{ii}\\
& D_i c_i^D = \sum_{j\in\mathcal{S}} t_{ji} - t_{ii}\\
& \sum_{i\in\mathcal{S}} U_i d_i^U = \sum_{i\in\mathcal{S}} D_i d_i^D\\
& 0 \le d_i^U \le M_i/U_i\\
& 0 \le d_i^U \le L\\
& 0 \le d_i^D \le L\\
& 0 \le c_i^U \le z\\
& 0 \le c_i^D \le z\\
& 0 \le t_{ij} (j \ne i), t_{ii} = 0
\end{aligned}$$

Note that the above convex problem (11) still embodies meaningful interpretation. Specifically, it assesses the benefit of an "ideal" condition under which each site distributes its *entire* intermediate data *arbitrarily* across the other sites, but rather than being constrained by the global task placement scheme. Intuitively, under this scenario, the data movement would make the most use of the bandwidth and price heterogeneities to aggressively reduce the total cost, by moving data out of the bottleneck sites with low bandwidth and high cost.

B. Two-stage Optimization to Handle the Uncertainty

Though the convex relaxation problem (11) can be readily solved using standard linear programming techniques, at the time when data movement begins, the query selectivity is unknown. On the other hand, solving the convex relaxation problem (11) only yields a feasible solution for input data movement, while we still need a solution for task placement. In response, following the philosophy of prediction-and-correction, we take a two-stage optimization approach to decouple task placement from the data movement.

Specifically, at the first stage when data is generated, we solve the convex relaxation problem with a predicted query selectivity to yield a valid solution for data placement. Then, at the second stage when the query arrives and the exact query selectivity is available, we correct the query response time and obtain the task placement solution by solving a separate optimization. The detailed algorithm is shown as follows:

Stage 1: when data is generated, we solve the convex relaxation problem (11) with the predicted selectivity $\hat{\alpha}$, and using the calculated d_i^U, d_i^D to do the input data movement. Albeit the duration of the shuffle phase (z) and the duration of intermediate data transfers (c_i^U and c_i^D) can be also obtained, they are invalid since the shuffle phase does not really happen. We let $T_i = D_i d_i^D - U_i d_i^U$, which will be used as the input of the second stage.

Stage 2: when query arrives, with the exact selectivity α and the input data movement finished in **Stage 1**, we solve the following linear problem to obtain the corresponding optimal task placement, and correct the duration of the shuffle phase, as well as the duration of the intermediate data transfers.

$$\begin{aligned}
\min \quad & z + \sum_{i\in\mathcal{S}} \{P_i^U U_i c_i^U + P_i^D D_i c_i^D\}, && (12)\\
\text{s.t.} \quad & \sum_{i\in\mathcal{S}} r_i = 1,\\
& U_i c_i^U = (1 - r_i)\alpha(M_i + T_i),\\
& D_i c_i^D = r_i \alpha(M - (M_i + T_i)),\\
& 0 \le c_i^U \le z,\\
& 0 \le c_i^D \le z,\\
& 0 \le r_i.
\end{aligned}$$

V. Performance Evaluation

In this section, we conduct trace-driven simulations to realisticly evaluate the proposed solution.

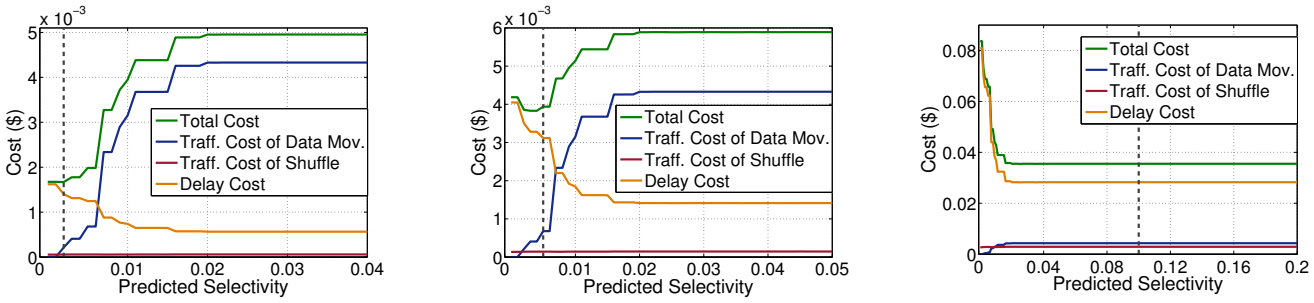
A. Experimental Setup

Trace: The trace we use for the simulations is a Hadoop cluster workload trace from Facebook [20]. The trace represents 24 hours in one day of 2009 on a cluster with 600 machines. The trace recorded the detailed information about the submission time, waiting time (the time interval from job submission to the beginning of map execution), map size, shuffle size, and reduce size of each job. By filtering out the jobs with zero map size or zero shuffle size, we get about 1446 jobs in 24 hours, *i.e.*, about 60 jobs are submitted to the cluster per hour in average.

Setup of the distributed edge sites: We mimic a distributed edge infrastructure with $S = 20$ sites. The up- and downlink bandwidths of each site are set to be random values between 50MB/s and 1GB/s. To mimic the cross-edge datasets, we split the total amount of input data of each job in the trace among the simulated 20 edge sites, by following a normal distribution as in [21]. Finally, we set the upload and download traffic prices (*i.e.*, P_i^U and P_i^D, respectively) at each link to be random values between 4×10^{-6}\$/MB and 7×10^{-6}\$/MB. Recall that in Sec. III-D, we have normalized the price of response time, θ to be 1.

B. Evaluation Results

To evaluate the performance of the two-stage joint optimization on data and task placement, proposed for single query in Sec. IV, we choose three benchmarks to compare with. The first benchmark is the aforementioned centralized approach that aggregates the data originated at different edge sites to a central datacenter, here we choose the site with the largest downlink bandwidth as the central datacenter. The second



(a) Actual selectivity $\alpha = 0.002$ (b) Actual selectivity $\alpha = 0.005$ (c) Actual selectivity $\alpha = 0.1$

Fig. 4. The total cost, as well as the three components: delay cost, traffic cost of data shuffle, and traffic cost of data movement verses the predicted selectivity, under different values of actual selectivity

benchmark is the "in place" approach that leaves the input data unmoved at split the reduce task proportional to the amount of input data originated at each site. The third benchmark is the approach that only optimize the task placement by solving the task placement problem (12) in Sec. IV-B.

To show how the algorithm works, we first pick one job from the trace to examine, the job is map-heavy, with a selectivity of $\alpha = 0.1$ and a map size of 3.02GB.

1) How does Prediction Error Affect: Recall that in the first stage when data is generated, the proposed algorithm moves the dataset based on a predicted selectivity. In order to show the effect of prediction error on the cost of jobs with different actual selectivity, we vary the actual selectivity in a relatively large range. Figure 4 depicts the effect of the predicted selectivity under various actual selectivity.

Specifically, Fig. 4(a) shows that when the actual selectivity $\alpha = 0.002$, the total cost firstly increases and then becomes stabilized as the predicted selectivity $\hat{\alpha}$ grows. However, Fig. 4(b) shows that when the actual selectivity $\alpha = 0.005$, the total cost firstly decreases and then increases and finally becomes stabilized as the predicted selectivity $\hat{\alpha}$ grows. Interestingly, we also find that when the predicted selectivity equals to the exact selectivity, then the total cost is larger than that with a under-predicted selectivity. The observation from Fig. 4(c) is also different from that of Fig. 4(a) and Fig. 4(b): the total cost decreases as the predicted selectivity $\hat{\alpha}$ grows, yet when the predicted selectivity equals to the exact selectivity, the total cost has became stabilized and also been minimized. Putting these three sub-figures together, we can conclude that, the prediction error does not show any common feature on the total cost of jobs with different actual selectivity, yet exact prediction of the selectivity may not necessarily minimize the total cost.

2) How does Query Lag Affect: Comparing with the third benchmark that solely optimizes the task placement, the joint optimization approach further optimizes input data distribution by moving input data within the lag between data generation and query arrival. Thus, it is intuitively that the improvement of the joint optimization approach (compared to the task optimization approach) largely depends on the length of the query lag. To validate this intuition, we vary the query lag and plot the total cost obtained by the joint optimization as well as

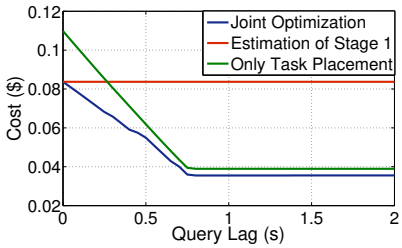


Fig. 5. The total cost brought by the joint optimization approach, task placement optimization approach, and predicted by the first-stage optimization, under varying lag

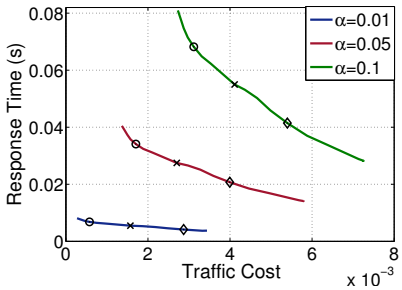


Fig. 6. Under various values of the actual selectivity, the response time diminishes and the traffic cost increases as the query lag increase. Circles, x-marks, and diamonds correspond to lag $L = 0.2,\ 0.4$, and 0.6, respectively.

the total cost predicted by the first-stage optimization, under perfect prediction at the first stage. From Fig. 5, we observe that both the total cost obtained by the joint optimization and predicted by the first-stage optimization firstly diminishes and then becomes stabilized. The rationale is that, when the query lag keeps relatively small, as the query lag increases, more input data movement can be performed, thus the total cost decreases as the lag expands. However, after the optimal amount of input data movement has been finished, no more input data would be moved as the lag increase, thus the cost becomes stabilized. Moreover, Fig. 5 also demonstrates that first-stage would over-estimate the total cost. This is in line with the enforcements that, in the first-stage optimization, each site uploads all the intermediate data to other sites.

When performing the joint optimization for different values of query lag, we record the response time and the total traffic cost (for both shuffle and input data movement) under

different lags and illustrate the former in Fig. 6. Clearly, as expected, with the increase of the query lag (meaning that the amount of input data been moved increases), the response time diminishes and the traffic cost rises. Further more, when comparing the curves of different selectivity, we can see that under the same lag, a larger selectivity would incur both increased response time and traffic cost. This is because that a larger selectivity means more intermediate data to be transferred, thus leads prolonged response time and rising traffic cost.

3) How Much Benefit of Joint Optimization: To assess the benefit of joint optimization on data and task placement, we repeat our algorithm as well as the three benchmarks for the 1446 jobs in the trace, here we assume that the first stage has perfect prediction of the query selectivity.

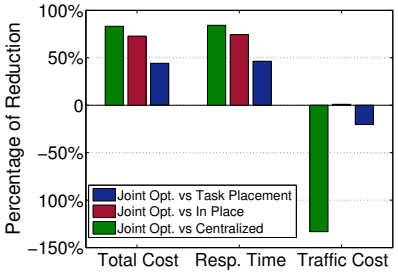


Fig. 7. Distribution of query response time under various scheduling policies

Fig. 7 depicts the reduction of the total cost, response time, and traffic cost brought by the joint optimization approaches versus the three benchmarks, under various prediction errors. Excitingly, we can see that the joint optimization on data and task placement can substantially reduce the total cost and response time when compared to the three benchmarks. Specifically, it can reduce the total cost by 83%, 72%, and 44%, and reduce the response time by 85%, 74%, and 46% when compared to the task placement optimization, in place approach, and centralized aggregation, respectively. However, we can also see that the joint optimization approach incurs 150% centralized approach, meaning that *cross-edge analytics may not necessarily reduce the WAN traffic cost*. Note that this is mainly due to the fact that the most of the jobs in the trace are shuffle-heavy, and the selectivity of these jobs can be as higher as 82.1, for these jobs, the traffic cost of intermediate data transfer can largely outperform the traffic cost of input data movement, thus aggregating the input data of these jobs can substantially reduce the traffic cost.

VI. Conclusion

In this paper, we study how to speed up cross-edge analytics at low traffic cost, via joint optimization on input data and task placement. To navigate the tradeoff between response time and traffic cost, we construct a unified cost objective to couple the above two conflicting sides. However, the resulted total cost minimization problem is rather challenging, due to its non-convexity and the uncertainty on the query selectivity. We show how these challenges can be addressed by a careful combination of convex relaxation and two-stage optimization. Extensive evaluations using a production trace from a Facebook cluster show that the two-stage joint optimization approach can reduce the total cost by up to 83% compared to the centralized aggregation approach.

Acknowledgements

This research was supported by National Key Research and Development Program under grant 2016YFB1000501. The corresponding author is Z. Zhou.

References

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, 2017.

[2] X. Chen, W. Li, S. Lu, Z. Zhou, and X. Fu, "Efficient resource allocation for on-demand mobile-edge cloud computing," *IEEE Transactions on Vehicular Technology*, vol. PP, no. 99, pp. 1–1, 2018.

[3] X. Chen, Z. Zhou, W. Wu, D. Wu, and J. Zhang, "Socially-motivated cooperative mobile edge computing," *IEEE Network*, vol. PP, no. 99, pp. 12–18, 2018.

[4] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *IEEE Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[5] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," in *Proc. of the IEEE/ACM IWQoS*, 2018.

[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proc. of USENIX OSDI*, 2004.

[7] 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 *Proc. of USENIX NSDI*, 2012.

[8] K. Kloudas, M. Mamede, N. Preguica, and R. Rodrigues, "Pixida: Optimizing data parallel jobs in wide-area data analytics," in *Proc. of VLDB*, 2015.

[9] 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.

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

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

[12] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: latency-aware video analytics on edge computing platform," in *Proceedings of the ACM/IEEE SEC*, 2017, pp. 15:1–15:13.

[13] G. Kar, S. Jain, M. Gruteser, F. Bai, and R. Govindan, "Real-time traffic estimation at vehicular edge nodes," in *Proceedings of the ACM/IEEE SEC*, 2017, pp. 3:1–3:13.

[14] 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.

[15] Measuring Internet Congestion: A Preliminary Report. https://ipp.mit.edu/sites/default/files/documents/Congestion-handout-final.pdf.

[16] Y. Chen, R. Mahajan, B. Sridharan, and Z. Zhang, "A Provider-side View of Web Search Response Time," in *Proc. of ACM SIGCOMM*, 2013.

[17] A. Singla, B. Chandrasekaran, P. Godfrey, and B. Maggs, "The Internet at the Speed of Light," in *Proc. of ACM Hotnets*, 2014.

[18] Z. Liu, M. Lin, A. Wierman, S. Low, and L. L. H. Andrew, "Geographical Load Balancing with Renewables," in *Proc. of ACM GreenMetrics*, 2011.

[19] S. Narayana, J. W. Jiang, J. Rexford, and M. Chiang, "To coordinate or not to coordinate? wide-area traffic management for data centers," in *Proc. ACM CoNEXT*, 2012.

[20] Facebook Cluster Trace. https://github.com/SWIMProjectUCB/SWIM/wiki/Workloads-repository.

[21] H. Xu and B. Li, "Joint Request Mapping and Response Routing for Geo-distributed Cloud Services," in *Proc. of IEEE INFOCOM*, 2013.