

Cost-Aware Big Data Processing across Geo-distributed Datacenters

Wenhua Xiao, Weidong Bao, Xiaomin Zhu, *Member, IEEE*, and Ling Liu, *Fellow, IEEE*

Abstract—With the globalization of service, organizations continuously produce large volumes of data that need to be analysed over geo-dispersed locations. Traditionally central approach that moving all data to a single cluster is inefficient or infeasible due to the limitations such as the scarcity of wide-area bandwidth and the low latency requirement of data processing. Processing big data across geo-distributed datacenters continues to gain popularity in recent years. However, managing distributed MapReduce computations across geo-distributed datacenters poses a number of technical challenges: how to allocate data among a selection of geo-distributed datacenters to reduce the communication cost, how to determine the VM (Virtual Machine) provisioning strategy that offers high performance and low cost, and what criteria should be used to select a datacenter as the final reducer for big data analytics jobs. In this paper, these challenges is addressed by balancing bandwidth cost, storage cost, computing cost, migration cost, and latency cost, between the two MapReduce phases across datacenters. We formulate this complex cost optimization problem for data movement, resource provisioning and reducer selection into a joint stochastic integer nonlinear optimization problem by minimizing the five cost factors simultaneously. The Lyapunov framework is integrated into our study and an efficient online algorithm that is able to minimize the long-term time-averaged operation cost is further designed. Theoretical analysis shows that our online algorithm can provide a near optimum solution with a provable gap and can guarantee that the data processing can be completed within pre-defined bounded delays. Experiments on WorldCup98 web site trace validate the theoretical analysis results and demonstrate that our approach is close to the offline-optimum performance and superior to some representative approaches.

Index Terms—Big Data Processing; Cloud Computing; Data Movement; Virtual Machine Scheduling; Online Algorithm.

1 INTRODUCTION

We are entering a big data era with more data generated and collected in a geographically distributed manner in many areas such as finance, medicine, social web, astronomy etc. With the increasing explosion of distributed data, the huge treasures hidden in it are waiting for us to explore for providing valuable insights. To illustrate, social web sites such as Facebook can uncover usage patterns and hidden correlations by analyzing the web site history records (e.g., click records, activity records et al.) to detect social hot event and facilitate its marketing decision (e.g., advertisement recommendation), and the Square Kilometre Array (SKA) [1], an international project to build the world's largest telescope distributed over several countries, need to fusion the geographically dispersed data for scientific applications. However, due to the properties such as large-scale volume, high complexity and dispersiveness of big data coupled with the scarcity of Wide-area bandwidth (e.g., trans-oceanic link), it is inefficient and/or infeasible to process the data with centralized solutions [2]. This has fueled strong companies from industry to deploy multi-datacenter cloud and hybrid cloud. These cloud technologies offer a powerful and cost-effective solution to deal with increasingly high velocity of big data generated from geo-distributed sources (e.g., Facebook, Google and Microsoft etc). For majority of the common organizations (e.g., SKA), it is economic to rent resource from public cloud, with considering the advantages of

cloud computing such as flexibility and pay-as-you-go business model.

MapReduce is a distributed programming model for processing large-scale dataset in parallel, which has shown its outstanding effectiveness in many existing applications [3], [4], [5]. Since original MapReduce model is not optimized for deployment across datacenters [6], aggregating distributed data to a single datacenter for centralized processing is a widely-used approach. However, waiting for such centralized aggregation suffers from significantly delays due to the heterogenous and limited bandwidth of user-cloud link. Notice that the bandwidth of inter-datacenter link is usually dedicated relatively high-bandwidth lines [7], moving the data to multiple datacenters for map operation in parallel and then aggregating the intermediate data to a single datacenter for reduce operation using inter-datacenter link has potential to reduce the latency. Furthermore, different kinds of cost (e.g., incurred by moving data or renting VM) also can be optimized considering the heterogeneity of the link speed, the dynamism of the data generation and the resource price. Therefore, distributing data from multi-sources into multi-datacenters and processing them using distributed MapReduce is an idea way to deal with the large volume dispersed data. Hitherto, the most important questions to be solved include: 1) how to optimize the placement of large-scale datasets from various locations onto geo-distributed datacenter cloud for processing and 2) how many resources such as computing resources should be provisioned to guarantee performance and availability while minimizing the cost. The fluctuation and multiple sources of generated data combined with the dynamic utility-driven pricing model of cloud resource make it a very challenging problem. The inter-dependency between multiple stages of distributed computation, such as the interplay between

Wenhua Xiao, Weidong Bao and Xiaomin Zhu are with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha, Hunan, P. R. China, 410073. E-mail: {wenhuaxiao, wdbao, xmzhu}@nudt.edu.cn

Ling Liu is with the college of computing at Georgia Institute of Technology, 266 Ferst Drive, Atlanta, GA 30332-0765, USA. E-mail: lingliu@cc.gatech.edu

the Map phase and the Reduce phase of MapReduce programs, further escalates the complexity of the data movement, resource provisioning and final reduce selection problems in geo-distributed datacenters.

In this paper, we address the problem of efficient scheduling with the goal of high performance, high availability and cost minimization by balancing five types of cost between the two MapReduce phases across multiple geo-distributed datacenters: bandwidth cost, storage cost, computing cost, migration cost, and latency cost.

Contributions: The major contributions of this work are summarized as follows:

- We propose a framework that can systematically handle the issues of data movement, resource provisioning as well as reducer selection under the context of running MapReduce across multiple datacenters, and VMs of different types and dynamic prices.
- We formulate the complex cost optimization problem as a jointed stochastic integer nonlinear optimization problem and solve it using Lyapunov optimization framework by transforming the original problem into three independent subproblems (data movement, resource provisioning and reduce selection) that can be solved with some simple solutions. We design an efficient and distributed online algorithm-*MiniBDP* that is able to minimize the long-term time-averaged operation cost.
- We formally analyze the performance of *MiniBDP* in terms of cost optimality and worst case delay. We show that the algorithm approximates the optimal solution within provable bounds and guarantees that the data processing can be completed within pre-defined delays.
- We conduct extensive experiments to evaluate the performance of our online algorithm with real world datasets. The experiments result demonstrate its effectiveness as well its superiority in terms of cost, system stability and decision-making time to existing representative approaches (e.g., the combinations of data allocation strategies (proximity-aware, load balance-aware) and the resource provisioning strategies(e.g., stable strategy, heuristic strategy).

The remainder of this paper is organized as follows: The next section reviews related work in the literature; Section 3 describes the system model and the problem formulation; Section 4 presents the online algorithm for solving the problem; The proposed algorithm is theoretically analyzed in section 5; Section presents the experiments and performance analysis using real-world trace. Section 7 concludes the paper with a summary and future work.

2 RELATED WORK

Computation models. MapReduce [3] is a popular and efficient distributed computing model that abstracts the data processing into two stages: Map and Reduce [6]. Extensions such as Twitter Storm [8] was proposed to handle real-time streaming data, Spark [4] was proposed as a solution that persistently keeps the distributed partitions in memory to eliminate disk I/O latency. To support data processing with evolving property, several efforts [9], [10] have added iterative or incremental support for MapReduce tasks. Recently, to deal with the issue that both data and compute

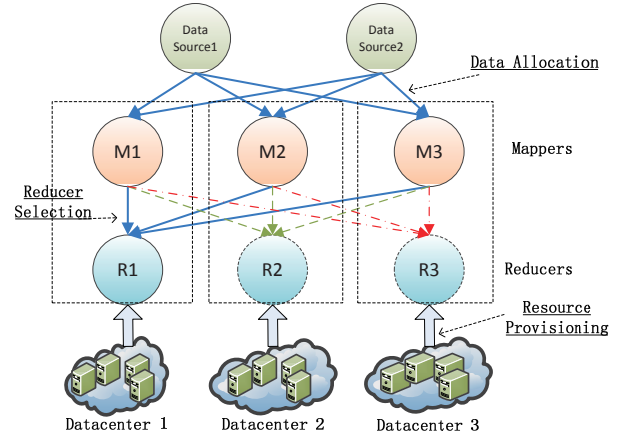


Fig. 1: Architecture of Big Data Processing with MapReduce Across Datacenters

resources are geo-distributed, the distributed MapReduce across datacenters was proposed [11], [12], [13]. To improve the efficiency of large-scale data processing, Sfrent et al. [14] proposed an asymptotic scheduling mechanism for many computing tasks for big data processing platforms. The common feature of these works is considering a static scenario where the data are pre-stored in the cloud and the amount of data are fixed.

Wide-Area Big-Data (WABD) analytics. Work on WABD has been a hot topic recently. Considering geo-dispersed data processing on clouds, Zhang et al. [7] proposed an online algorithm to migrate dynamically generated data from various locations to the clouds and studied how to minimize the bandwidth cost of transferring data for delay-tolerant processing with multiple Internet Service Providers (ISPs) [15]. Zhang et al. [16] studied how to efficiently schedule and perform analysis over data that is geographically distributed across multiple datacenters and designed system-level optimizations including job localization, data placement and data pre-fetching for improving performance of Hadoop service provisioning in a geo-distributed cloud. Targeting at query analytics over geo-distributed datacenters, studies focus on different goals (e.g., either reducing bandwidth cost [2], [17], [18] or execution response time [19]). *Geode* [2] is proposed to solve the problem of querying wide-area distributed data with goal of reducing bandwidth cost, but it makes no attempt to minimize execution latency and does not support general computations task that go beyond SQL query under MapReduce framework. *WANAlytics* [17] is designed for arbitrary computation with DAGs of task and proposed a heuristic algorithm to optimize tasks execution as well as an intermediate data caching strategy to reduce bandwidth cost. *PIXIDA* [18] is proposed to minimize the traffic incurred from data movement across resource constrained links. In contrast to *MiniBDP*, it formulates the traffic minimization optimization into a graph partitioning problem. *Iridium* [19] is the closest work since it also optimizes the data and task placement to achieve the goal of minimizing the response time of query analysis across geo-distributed sites. However, its approach is rather different from *MiniBDP* since it needs to estimate the query arrivals and ignores the CPU and storage cost. In addition, *MiniBDP* shows delay bounds while *Iridium* does not.

Management of multiple datacenters. Managing multiple geographically distributed datacenters has attracted companies such as Facebook, Google, HP and Cisco. To support geo-distributed

hadoop data storage, Facebook developed a project Prism [20] by adding a logical abstraction layer to Hadoop cluster. Focusing on fault tolerance and load balancing, Google deployed its database system Spanner [21] in a distributed manner, which is able to automatically migrating data across datacenters. HP [22] and Cisco [23] have also made efforts to manage their geo-distributed datacenters by optimizing the inter-datacenters network on the layer of data link. However, current practical methods are limited by their transport dependency, complexity and lack of resilience. Further, these methods mainly focus on providing better service quality for increasingly global user demands but not on data computations.

Recently, Lyapunov optimization technique was [24] applied to cloud computing context to deal with job admission and resource allocation problem [25], [26]. Yao et al. [27] extends it from the single time scale to two-time-scale for achieving electricity cost reduction in geographically distributed datacenters. Besides, this approach is used for resource management in cloud-based video service [28], [29]. In this paper, we apply this technique to address the issue of data moving and resource provisioning for big data processing in cloud with geo-distributed MapReduce.

To summarize, differs to aforementioned studies, our goal is to minimize overall cost when processing geo-dispersed big data across multiple datacenters, by balancing computation cost, bandwidth cost, storage cost, migration cost and latency cost, not only one or part of them. Further, we incorporate dynamic resource provisioning into the framework and make decision on the data movement, resource provisioning and reducer selection simultaneously at a long run. In addition, we consider the problem on the granularity of Map and Reduce as well as the data flow between the two phases that support incremental style across distributed datacenters.

3 MODELING AND FORMULATION

In this section, we first introduce the preliminary knowledge on MapReduce and the execution path of data over geo-distributed datacenters, and then we present the system model.

3.1 Preliminaries

In MapReduce model, Mapper process the input datasets and output a set of $\langle key, value \rangle$ intermediate pairs at Map phase, while Reducer receive all the intermediate data from mappers and merge the values according to a specific key to produce smaller sets of values at Reduce phase. Both of them can be deployed in different nodes.

Under the environment of distributed datacenters, the execution path of geo-distributed data is of particular importance. As concluded by Chamikara et.al. [12], there are three execution paths for data processing with MapReduce across datacenters: *COPY*, *MULTI* and *GEO*. *COPY* is a strategy that copies all the sub-datasets into a single datacenter before handing them with MapReduce. However, it is inefficient when the output data generated by MapReduce is much smaller than the inputs. *MULTI* is a strategy that executes MapReduce job separately on each sub-dataset and then aggregates the individual results. The drawback of this strategy lies in that the expected outcome is yielded only if the order of the MapReduce jobs does not have an impact on the final result. *GEO* is a strategy that executes the Map operation in different datacenters and then copy the intermediate data to a single datacenter for Reduce operation. This

is suitable for those applications where the jobs are correlated in the Reduce phase, e.g., determining the median size of the pages in a Web cache, or those applications where the intermediate data is smaller than the input. As reported in [13], by measuring the Hadoop traces of about 16000 jobs from Facebook, there are about 70% of jobs whose input data is larger than the corresponding intermediate data. Therefore, *GEO* conducts the map operation in each datacenter and then aggregates the intermediate data into a single datacenter will reduce cross-region bandwidth cost. Based on above consideration, we consider the *GEO* execution path in problem modelling.

3.2 System Model

Without loss of generality, we consider such a system scenario where a DSP (Data Service Provider) manages multiple data sources and transfers all the data into cloud for processing using MapReduce. The DSP may either deploy its private datacenters (e.g., Google deploys tens of datacenters over the world) or rent the resource from public clouds (e.g., SKA may rent the resource from public cloud such as Amazon EC2). Specially, for the DSP that have its private cloud, datasources overlaps datacenters since generated data are collected and stored in its own datacenters. System architecture is presented in Fig. 1: Data sources from multiple geographical data locations continuously produce massive data. Data analysis applications are deployed in the cloud and the data sources is connected to datacenters located in multiple places. In this model, data are moved to the datacenters once they are generated and are processed in a incremental style in which only the newly arrived data are computed and the intermediate data from past can be reused. Specifically, both mappers and reducers are running on every datacenter. As the *GEO* execution path mentioned above is considered in this paper, there are two corresponding phases for the data moving procedure. At the first phase, data can be moved to any datacenter for Map operation. At the second phase, the intermediate data of Mappers must be moved into a single datacenter with consideration of data correlations. As shown in Fig. 1, the bold line is an example of execution path, which shows that the raw data from data source 1 and data source 2 are moved to multiple datacenters for Map operation and then the output data of Mappers are aggregated into the Reducer in datacenter 1 for Reduce operation.

Formally, let \mathcal{D} be the set of geographically distributed datacenters with size of $D = |\mathcal{D}|$ (indexed by $d(1 \leq d \leq D)$) and \mathcal{K} be the set of VM types with size $K = |\mathcal{K}|$, each of which has a specific capacity v_k with configurations such as CPU and memory. All types of VMs can be provisioned in each datacenter. Data are dynamically and continuous generated from $R = |\mathcal{R}|$ different datasource locations (indexed by $r, 1 \leq r \leq R$), denoted as a set \mathcal{R} . Data from any location can be moved to any datacenter for Map operation and then aggregate the intermediate data into a single datacenter. To be realistic, we assume that the bandwidth B_{rd} from data location r to datacenter d is limited. Also note that inter-datacenter links (e.g., trans-oceanic links) are expensive to lay down, so the costs of using these links are considered as a first-order entity when migrating the intermediate data among datacenters. In addition, the data generation in each location is independent and the prices of the resource (e.g., VM) in each datacenter are varied in both spatial and temporal domain.

The system runs according to time slots, which is denoted by $t = 0, 1, \dots, T$. In each time slot, the DSP needs to make

TABLE 1: IMPORTANT NOTATIONS

\mathcal{D}	set of datacenters (DC)
\mathcal{R}	set of data locations
\mathcal{K}	set of VM types
$a_r(t)$	amount of the data generated from datasource (DS) r at t
A_{max}^r	max amount of data generated from DS r
$\lambda_r^d(t)$	amount of the data allocated to d from DS r at t
$N_d^{k,max}$	max number of VMs of type- k in DC d
$m_d^k(t)$	number of type- k VM provisioned for Map in DC d at t
$n_d^k(t)$	number of type- k VM provisioned for Reduce in DC d at t
$p_d^k(t)$	price of type- k VM in DC d at t
s_d	price of storage in datacenter d
b_r^d	price of bandwidth between DS r and DC d
U_r^d	uplink bandwidth between DS r and DC d
L_r^d	the latency between DS r and DC d
v_k	data processing rate of type- k VM
ϵ_d	preset constant for controlling queueing delay in $M_d(t)$
σ_d	preset constant for controlling queueing delay in $R_d(t)$
l	max delay of data process
$M_d(t)$	unprocessed data at Map phase in DC d at t
$R_d(t)$	unprocessed data at Reduce phase in DC d at t
$Y_d(t)$	virtual queue associate with $M_d(t)$ to guarantee its delay
$Z_d(t)$	virtual queue associate with $R_d(t)$ to guarantee its delay

the decision about moving how much data from data location r to datacenter d , renting how many resources to support its data processing from each datacenter, and selecting which datacenter for Reduce operation. Our goal therefore is to minimize the overall cost of big data analysis in clouds while guaranteeing the delay in the long run. For ease of reference, important notations are summarized in Table 1.

3.3 Problem Formulation

In this subsection, based on the system model aforementioned, we formulate the problem mathematically as follows.

Decision variables. The three decisions to be made are:

(1) Data allocation variable: $\lambda_r^d(t)$, denotes the amount of the data allocated to d from data location r at t , which means that the data generated from each location can be moved to any datacenter for analysis. Let $a_r(t)$, A_{max}^r , U_r^d be the amount of data generated from the r -th region at time slot t , the max volume of data generated in location r and the upload capacity between region r and datacenter d , respectively. Hence, we have:

$$a_r(t) \leq A_{max}^r, \forall r \in \mathcal{R}, t \in [1, T], \quad (1)$$

$$a_r(t) = \sum_{d \in \mathcal{D}} \lambda_r^d(t), \forall r \in \mathcal{R}, t \in [1, T], \quad (2)$$

$$0 \leq \lambda_r^d(t) \leq U_r^d, \forall r \in \mathcal{R}, d \in \mathcal{D}, t \in [1, T], \quad (3)$$

where Eq.(2) ensures that the sum of data allocated to each datacenter at one time slot is equal to the total amount data generated at that time slot. Eq.(3) ensures that the total amount of data uploaded via link $\langle r, d \rangle$ should not exceed the upload capacity of link $\langle r, d \rangle$. The variable set is denoted as $\lambda(t) = \{\lambda_r^d(t), \forall r \in \mathcal{R}, \forall d \in \mathcal{D}\}$.

(2) VM provisioning variable: $m_d^k(t)$, $n_d^k(t)$, $\forall d \in \mathcal{D}, \forall k \in \mathcal{K}$, denote the number of type- k VM rented from datacenter d at time slot t for Map and Reduce operation, respectively. They can be scaled up and down over time slots. Since the computation resource in a datacenter is limited, we let $N_d^{k,max}$ be the max number of type- k VM in datacenter d . Therefore, we have:

$$0 \leq n_d^k(t) + m_d^k(t) \leq N_d^{k,max}, \forall d, \forall k, t \in [1, T], \quad (4)$$

which means that the amount of resource employed by Map and Reduce operation cannot surpass the available resources in a specific datacenter. We denote $\mathbf{m}(t) = \{m_d^k(t), \forall d \in \mathcal{D}, \forall k \in \mathcal{K}\}$. $\mathbf{n}(t)$ is defined similarly.

(3) Reducer selection variable: $x_d(t), \forall d \in \mathcal{D}$. Since all the intermediate data from mappers will be aggregated into only one datacenter for Reduce operation at time slot t , $x_d(t)$ needs to be defined as a binary variable. It indicates whether datacenter d is the target datacenter to execute Reduce operation at time slot t ($x_d(t) = 1$) or not ($x_d(t) = 0$). Formally, we have:

$$\sum_{d \in \mathcal{D}} x_d(t) = 1, x_d(t) \in \{0, 1\}, \forall t \in [1, T], \quad (5)$$

where $\sum_{d \in \mathcal{D}} x_d(t) = 1$ ensures that there is only one datacenter running Reducer at time slot t . We defined set $\mathbf{x}(t) = \{x_d(t), \forall d \in \mathcal{D}\}$.

Cost. The goal of the DSP is to minimize the overall cost incurred in the system by optimizing the amount of data allocated to each datacenter, the number of resources needed, and the suitable datacenter for Reduce operation. Specifically, the following cost components are considered in this paper: bandwidth cost, storage cost, latency cost, computing cost and migration cost.

(1) Bandwidth cost, storage cost and the latency cost. Usually, the bandwidth price is varied over different VPN links because they often belong to different Internet service providers. Let b_r^d be the price of transferring 1 GB data between data location $r \in \mathcal{R}$ and datacenter $d \in \mathcal{D}$, then the bandwidth cost of moving data into cloud at t is: $\sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} \lambda_r^d(t) \cdot b_r^d$. For storage cost, which is an important factor to be considered in choosing the datacenter for data analysis due to large amount of data for big data application. Let s_d , $W_d(t)$ represent the price of data storing and the amount of unprocessed data in datacenter $d \in \mathcal{D}$ respectively, then the storage cost at t is: $\sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} \lambda_r^d(t) \cdot s_d + \sum_{d \in \mathcal{D}} W_d(t) \cdot s_d$. In particular, it can be obtained that $W_d(t) = M_d(t) + R_d(t)$ from (17) and (19). The latency incurred by uploading data to the datacenters is also an important performance measure, which is to be minimized in the data moving process. Let L_r^d denote the latency between the data location $r \in \mathcal{R}$ and the datacenter $d \in \mathcal{D}$. These delays are mainly determined by the respective geographic distance and bandwidth of links. As suggested in [7], we convert the latency into monetary cost. Therefore, we can define the latency cost as: $\sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} \alpha \cdot \lambda_r^d(t) \cdot L_r^d$, where α is a weight converting latency into a monetary cost. Therefore, the total cost of this part can be defined as:

$$C_{sbl}(\lambda(t)) = \sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} \lambda_r^d(t) \cdot (s_d + b_r^d + \alpha L_r^d) + \sum_{d \in \mathcal{D}} W_d(t) \cdot s_d. \quad (6)$$

(2) Computing cost. Due to the variance of VM price over time slots, the number of the VMs rented from datacenter has important impact on the overall cost of the system as well as QoS of the big data application. Let $p_d^k(t)$ be the price of type- k VM in datacenter d at time slot t , which is diverse in both spatial and time space. Then the computing cost can be calculated as follows,

$$C_p(\mathbf{m}(t), \mathbf{n}(t)) \triangleq \sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} \{m_d^k(t) + n_d^k(t)\} \cdot p_d^k(t). \quad (7)$$

(3) Migration cost. In many applications, analyzing data not only uses the data at current time but also needs the historical

data (e.g., incremental analytics re-use the previous computation results rather than recompute them when new data arrives [9]). Therefore, the intermediate data including the historical ones generated by Map operations in other geo-distributed datacenters will be transferred to the selected reducer, incurring a migration cost. Without loss of generality, assuming the historical data from μ previous time slots will be reused, we denote the amount of historical data to be transferred from datacenter i at t is $h_i(t) = \sum_{\tau=t-\mu}^{t-1} \beta_\tau f_i(\tau)$, where $f_i(\tau)$ represents the amount of intermediate data generated in datacenter i at time τ . $f_i(\tau)$ can be estimated using the data processed at Map phase at time τ since there is a ratio (e.g., γ) between the raw data size and the intermediate data size for a specific application [30]. $\beta_\tau \in [0, 1]$ indicates the ratio of historical data to be transferred. In particular, it satisfies $\beta_a < \beta_b$ if $a < b$, which means the importance of the historical data declines as time goes by. The weight can be determined by the specific application. Furthermore, we denote $\Phi_{id}(\cdot)$ as a non-decreasing migration cost function (including bandwidth cost and latency cost) for moving data from datacenter i to datacenter d , which can be defined based on the bandwidth prices and the geographic distances among datacenters. Also, it satisfies $\Phi_{id}(\cdot) = 0$ when $i = d$ because it is unnecessary to transfer data within the same datacenter. Hence, the total migration cost incurred in the system at time slot t is:

$$C_{mgr}(\mathbf{m}(t), \mathbf{x}(t)) \triangleq \sum_{d \in \mathcal{D}} \left\{ x_d(t) \sum_{i \in \mathcal{D}} \Phi_{id}(h_i(t)) \right\}. \quad (8)$$

Based on above cost formulations, the overall cost incurred in the system at time slot t can be calculated as:

$$C(\mathbf{m}(t), \mathbf{n}(t), \lambda(t), \mathbf{x}(t)) = C_p(\mathbf{m}(t), \mathbf{n}(t)) + C_{sbl}(\lambda(t)) + C_{mgr}(\mathbf{m}(t), \mathbf{x}(t)). \quad (9)$$

Objective. The problem of minimizing the time-average cost of data moving and processing within a long-term period $[0, T]$ can be formulated as:

$$\mathbf{P1.} \quad \min : \quad \bar{C} \quad (10)$$

$$\text{s.t. :} \quad 0 \leq \lambda_r^d(t) \leq U_r^d, \forall r, \forall d, t \in [1, T]; \quad (11)$$

$$a_r(t) = \sum_{d \in \mathcal{D}} \lambda_r^d(t), \forall r, t \in [1, T]; \quad (12)$$

$$0 \leq n_d^k(t) + m_d^k(t) \leq N_d^{k, \max}, \forall d, \forall k, t \in [1, T]; \quad (13)$$

$$m_d^k(t) \in Z^+ \cup 0, n_d^k(t) \in Z^+ \cup 0, \forall d, \forall k, t \in [1, T]; \quad (14)$$

$$\sum_{d \in \mathcal{D}} x_d(t) = 1, x_d(t) \in \{0, 1\}, \forall t \in [1, T]; \quad (15)$$

$$\bar{\lambda}_d \leq \bar{m}_d, \bar{F}_d \leq \bar{n}_d, \forall d; \quad (16)$$

where $\bar{C} \triangleq \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^{T-1} C(\mathbf{m}(t), \mathbf{n}(t), \lambda(t), \mathbf{x}(t))$. $\bar{\lambda}_d$ is the time-averaged data size allocated to datacenter d and \bar{m}_d is time-averaged VM resource provisioned for Map phase at datacenter d . \bar{F}_d represents the average intermediate data size input to Reduce phase and \bar{n}_d is time-averaged VM resource provisioned for Reduce phase at datacenter d . Thus, the constraint (16) guarantees the stability of Map workload queue M_d and Reduce

workload queue R_d , by ensuring that the arrival data rate is no higher than the average process rate.

As the data generation is stochastic, x is an integer constrained variable and $h_i(t)$ is a nonlinear function, it can be easily verified that the problem is a constrained stochastic integer nonlinear optimization problem and our objective is to minimize the long-term average cost by optimizing the amount of data allocated to each datacenter, the number of VMs rented from the datacenters as well as selecting the optimal reducer. However, it is often infeasible to solve the problem efficiently in a centralized solution when T is large. To deal with this complex problem, we employ a recent developed optimization technique-Lyapunov optimization [31] as shown in the section 4.

4 ONLINE ALGORITHM DESIGN

An outstanding feature of Lyapunov optimization is that it does not need future information about workload. By greedily minimizing the drift-plus-penalty at each time slot, it can solve the long-term optimization problem efficiently with a solution that can be proved to arbitrarily close to the optimum. Next, we first transform the problem **P1** to an optimization problem of minimizing the Lyapunov drift-plus-penalty term and then design the corresponding online algorithm.

4.1 Problem Transformation

Queues Design. As the incremental data processing style is considered in the paper, we model the data processing evolution as a queue model. In each datacenter, to describe the two data processing procedures (Map and Reduce) running in the system, we design the corresponding queues as follows.

(1) For the Map phase, let $M_d(t)$ be the amount of unprocessed data in Map queue in datacenter d at time slot t . The queue is initialized as $M_d(0) = 0$, and then the update of the queue $M_d(t)$ can be described as follows:

$$M_d(t+1) = \max[M_d(t) - \sum_{k \in \mathcal{K}} m_d^k(t) \cdot v_k, 0] + \sum_{r \in \mathcal{R}} \lambda_r^d(t). \quad (17)$$

The above queue evolution implies that the amount of processed data and newly-arrived data at Map phase are $\sum_{k \in \mathcal{K}} m_d^k(t) \cdot v_k$ and $\sum_{r \in \mathcal{R}} \lambda_r^d(t)$, respectively.

To guarantee that the worst-case queuing delay in queue $M_d(t)$, $\forall d \in \mathcal{D}$, is bounded by the max Map workload delay l_m , we design a related virtual queue $Y_d(t)$ according to the ϵ -persistent service technique for delay bounding in [32]. Similarly, the backlog of virtual queue $Y_d(t)$ is initialized as $Y_d(0) = 0$, then it is updated as follows:

$$Y_d(t+1) = \max[Y_d(t) + 1_{M_d(t)>0}(\epsilon_d - \sum_{k \in \mathcal{K}} m_d^k(t) \cdot v_k) - 1_{M_d(t)=0} \sum_{k \in \mathcal{K}} N_d^{k, \max} \cdot v_k, 0], \quad (18)$$

where the indicator function $1_{M_d(t)>0}$ equals to 1 when $M_d(t) > 0$, and 0 otherwise. Similarly, $1_{M_d(t)=0}$ equals to 1 when $M_d(t) = 0$, and 0 otherwise. ϵ_d is a preset constant that can be used to control the bound of delay for Map queue. It can be proved that we are able to guarantee all data being processed with delays at most l_m time slots if the length of $M_d(t)$ and $Y_d(t)$ over time slots can be guaranteed. It is also proved that l can be set

as $l_m = [(M_d^{\max} + Y_d^{\max})/\epsilon_d]$, where M_d^{\max} and Y_d^{\max} are the bound of queues $M_d(t)$ and $Y_d(t)$ respectively. Details can be seen in Theorem 5.3.

(2) For the Reduce phase, similar to the Map phase, the corresponding queue in datacenter d is denoted as $R_d(t)$ with $R_d(0) = 0$, and the updating of this queue can be calculated as follows.

$$R_d(t+1) = \max[R_d(t) - \sum_{k \in \mathcal{K}} n_d^k(t)v_k, 0] + x_d(t) \cdot F_d(t), \quad (19)$$

where $F_d(t) = \sum_{\tau=t-\mu}^{t-1} (\beta_\tau \sum_{i \in \mathcal{D}} f_i(\tau))$ is the amount of intermediate data from other datacenters, including the historical data of past μ time slots. From the above equations, we can know that it admit only part of the data generated within the same time slot is processed. So do moving its intermediate data to a reducer. In reality, the system will wait for all the intermediate data to produce the final result.

Accordingly, its virtual queue can be defined as:

$$Z_d(t+1) = \max[Z_d(t) + 1_{R_d(t) > 0}(\sigma_d - \sum_{k \in \mathcal{K}} n_d^k(t) \cdot v_k) - 1_{R_d(t) = 0} \sum_{k \in \mathcal{K}} N_d^{k, \max} \cdot v_k, 0]. \quad (20)$$

In theory, the worst case delay of queue $R_d(t)$ can also be guaranteed as shown in Theorem 5.3.

Problem Transformation. Let $\mathbf{M}(t) = [M_d(t)]$, $\mathbf{Y}(t) = [Y_d(t)]$, $\mathbf{R}(t) = [R_d(t)]$ and $\mathbf{Z}(t) = [Z_d(t)]$, $\forall d \in \mathcal{D}$ denote the matrix of Map queues and Reduce queues respectively. Then, to measure the congestion of data processing procedure, we use $\Theta(t) = [\mathbf{M}(t); \mathbf{Y}(t); \mathbf{R}(t); \mathbf{Z}(t)]$ to denote the combined matrix of Map queues and Reduce queues. Thus, the Lyapunov functions can be defined as follows:

$$L(\Theta(t)) = \frac{1}{2} \sum_{d \in \mathcal{D}} \{M_d(t)^2 + Y_d(t)^2 + R_d(t)^2 + Z_d(t)^2\}, \quad (21)$$

where $L(\Theta(t))$ measures the queue backlogs in the system. Furthermore, to keep the stability of above queues by persistently pushing the Lyapunov function to a low congestion state, the one-slot Lyapunov drift is introduced as:

$$\Delta(\Theta(t)) = \mathbb{E}\{L(\Theta(t+1)) - L(\Theta(t)) | \Theta(t)\}. \quad (22)$$

According to the Lyapunov optimization framework, the drift-plus-penalty, which balances the queue stability and system cost, can be obtained by adding the cost incurred by the system to the above Lyapunov drift, namely,

$$\Delta(\Theta(t)) + V \cdot \mathbb{E}\{C(\mathbf{m}(t), \mathbf{n}(t), \lambda(t), \mathbf{x}(t)) | \Theta(t)\}, \quad (23)$$

where V is a non-negative weight that affects the balance between the cost optimization and drift minimization. Intuitively, a larger V will causes a smaller cost, and vice versa. Therefore, the problem **P1** can be transformed into problem **P2** as following:

$$\mathbf{P2.} \min : \quad (23) \quad (24)$$

$$\text{s.t. :} \quad (11)(12)(13)(14)(15). \quad (25)$$

To solve problem **P2**, rather than directly minimizing the drift-plus-penalty expression (23), we seek to minimize the upper bound for it, without undermining the optimality and performance of the algorithm according to [31]. The key point of the problem

is therefore to find an upper bound on the expression(23). It can be proved that, under any decision strategy, the expression (23) satisfies:

$$\begin{aligned} & \Delta(\Theta(t)) + V \cdot \mathbb{E}\{C(\mathbf{m}(t), \mathbf{n}(t), \lambda(t), \mathbf{x}(t)) | \Theta(t)\} \\ & \leq B \\ & + \mathbb{E}\left\{\sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} m_d^k(t) \cdot (Vp_d^k(t) - M_d(t)v_k - Y_d(t)v_k) | \Theta(t)\right\} \\ & + \mathbb{E}\left\{\sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} n_d^k(t) \cdot (Vp_d^k(t) - R_d(t)v_k - Z_d(t)v_k) | \Theta(t)\right\} \\ & + \mathbb{E}\left\{\sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} \lambda_r^d(t) \cdot (Vs_d + Vb_r^d + V\alpha L_r^d + M_d(t)) | \Theta(t)\right\} \\ & + \mathbb{E}\left\{\sum_{d \in \mathcal{D}} x_d(t) \left\{V \sum_{i \in \mathcal{D}} \Phi_{id}(h_i(t)) + R_d \sum_{i \in \mathcal{D}} h_i(t)\right\} | \Theta(t)\right\} \end{aligned} \quad (26)$$

where $B = 2 \sum_{d \in \mathcal{D}} (\sum_{k \in \mathcal{K}} N_d^{k, \max} v_k)^2 + \frac{1}{2} \sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} \lambda_r^{\max} + \frac{1}{2} \sum_{d \in \mathcal{D}} ((\epsilon_d)^2 + (\sigma_d)^2) + \frac{1}{2} D(\mu\beta_{\max} \sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} N_d^{k, \max} v_k)^2$. Detailed proof please refer to [31].

4.2 Online Control Algorithm Design

Fortunately, by investigating the R.H.S (Right Hand Side) of (26), we can equivalently decouple the problem into three subproblems: 1) data allocation, 2) resource provisioning and (3) reducer selection. The details of the solution for the above three subproblems are given as follows.

1) Data Allocation: On the R.H.S of (26), a carefully observation on the relationship among different variables reveals that the data allocation problem can be written as follows:

$$\mathbb{E}\left\{\sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} \lambda_r^d(t) \cdot (Vs_d + Vb_r^d + V\alpha L_r^d + M_d(t)) | \Theta(t)\right\}. \quad (27)$$

Furthermore, since the data generated at each datasource are independent, the centralized optimization can be implemented independently and distributedly at each datasource. Considering the data allocation in datasource r at time t , we should solve the following problem.

$$\begin{aligned} & \min \sum_{d \in \mathcal{D}} \lambda_r^d(t) [Vs_d + Vb_r^d + V\alpha L_r^d + M_d(t)] \\ & \text{s.t.} (11)(12) \end{aligned} \quad (28)$$

In fact, problem (28) can be regarded as a generalized min-weight problem in which the amount of data from datasource r moved to datacenter d ($\lambda_r^d(t)$) is weighted by the queue backlog $M_d(t)$, bandwidth price b_d , storage price s_d and the latency cost $L(r, d)$. Intuitively, the data inclines to be allocated to the datacenter with the minimal value of weight $[Vs_d + Vb_r^d + V\alpha L_r^d + M_d(t)]$. Note the allocation variable $\lambda_r^d(t)$ is constrained by the uplink capacity (i.e., $0 \leq \lambda_r^d(t) \leq U_r^d$), we can allocate the data to the datacenters within their uplink capacities according to their weight order. The detailed algorithm for solving this problem can be seen in Algorithm 1. Obviously, the complexity of algorithm 1 is with $\mathcal{O}(D \times R)$, thus the averaged complexity of each element is with $\mathcal{O}(1)$. Obviously, the strategy exhibits that all the left data can be allocated if the data to be moved (i.e., a_{left}) is less than the corresponding uplink capacity. Otherwise, the amount of data equals to the uplink capacity (i.e., U_r^d) will be allocated. Repeating the procedures above, all the data can be allocated until there is no left data.

Algorithm 1: The Algorithm of Solving $\lambda_r^d(t)$

```

1 Input:  $\alpha, V, L_r^d, M_d(t), a_r(t), s_d, b_r^d, U_r^d (\forall d \in \mathcal{D}, \forall r \in \mathcal{R})$ 
2 Output:  $\lambda_r^d(t)$ 
3 foreach  $r \in \mathcal{R}$  do
4   Sort  $\mathbf{Q}_r$  with ascending order where
    $Q_r^d = V s_d + V b_r^d + V \alpha L_r^d + M_d(t)$ ;
5   Initialize  $a_{left} = a_r(t)$ ;
6   while  $\mathbf{Q}_r \neq \emptyset \cap a_{left} > 0$  do
7      $d = \text{head}(\mathbf{Q}_r)$ ;
8     if  $U_r^d \geq a_{left}$  then
9        $\lambda_r^d(t) = a_{left}$ ;
10    else
11       $\lambda_r^d(t) = U_r^d$ ;
12     $a_{left} = a_{left} - \lambda_r^d(t)$ ;

```

2) Resource Provisioning: The part related to variable $m_d^k(t)$ and $n_d^k(t)$ in the R.H.S of (26) can be regarded as resource provisioning problem if we remove the constant term. Therefore, the optimal VM provisioning strategy can be obtained by solving the following problem:

$$\begin{aligned}
& \min \mathbb{E} \left\{ \sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} m_d^k(t) (V p_d^k(t) - M_d(t) v_k - Y_d(t) v_k) | \Theta(t) \right\} \\
& + \mathbb{E} \left\{ \sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} n_d^k(t) (V p_d^k(t) - R_d(t) v_k - Z_d(t) v_k) | \Theta(t) \right\} \\
& \text{s.t. (13)(14)}
\end{aligned} \tag{29}$$

As the resource provisionings in each datacenter are independent, similar to data allocation, problem (29) also can be solved independently and distributedly within each datacenter. Considering the resource problem within datacenter d , we can rewrite it as (30).

$$\begin{aligned}
& \min \mathbb{E} \left\{ \sum_{k \in \mathcal{K}} m_d^k(t) (V p_d^k(t) - M_d(t) v_k - Y_d(t) v_k) | \Theta(t) \right\} \\
& + \mathbb{E} \left\{ \sum_{k \in \mathcal{K}} n_d^k(t) (V p_d^k(t) - R_d(t) v_k - Z_d(t) v_k) | \Theta(t) \right\} \\
& \text{s.t. (13)(14)}
\end{aligned} \tag{30}$$

Using the basic knowledge of linear programming, the solution to the above linear problem can be derived as shown in Eq.(31)(refer to next page), which indicates that a type- k VM is preferred to be rented in t when its price $p_d^k(t)$ is small, and the VM whose capacity, v_k , is large is more likely to be rented too.

3) Reducer Selection: The part related to variable $x_d(t)$ in the R.H.S of (26) is therefore considered as the reducer selection problem. It can be written as follows.

$$\begin{aligned}
& \min \mathbb{E} \left\{ \sum_{d \in \mathcal{D}} x_d(t) \left\{ V \sum_{i \in \mathcal{D}} \Phi_{id}(h_i(t)) + R_d \sum_{i \in \mathcal{D}} h_i(t) \right\} | \Theta(t) \right\} \\
& \text{s.t. (15)}
\end{aligned} \tag{32}$$

Note that $h_i(t) = \sum_{\tau=t-\mu}^{t-1} \beta_\tau f_i(\tau)$ is known since $f_i(\tau), \tau \in [t-\mu, t-1]$ is known at time slot t , it also becomes a min-weight problem. Hence, it can be easily derived that:

$$x_d(t) = \begin{cases} 1, & d = d^* \\ 0, & \text{else} \end{cases}, \tag{33}$$

Algorithm 2: Procedures of the Algorithm *MiniBDP*

```

1 Input:
2  $M_d(t), Y_d(t), R_d(t), Z_d(t), a_r(t), v_k, s_d, b_r^d, L_r^d, \mu, \alpha, \beta, \gamma$ 
 $N_d^{k, \max}, A_{\max}^r, p_d^k(t), V, \alpha (\forall d \in \mathcal{D}, \forall r \in \mathcal{R}, \forall k \in \mathcal{K})$ 
3 Output:
4  $m_d^k(t), n_d^k(t), \lambda_r^d(t), x_d(t) (\forall d \in \mathcal{D}, \forall r \in \mathcal{R}, \forall k \in \mathcal{K})$ 
5 Resource provisioning:
6 foreach datacenter  $d \in \mathcal{D}$  do
7   Get the VM provisioning strategies for Map ( $m_d^k(t)$ ) and
   Reduce ( $n_d^k(t)$ ) by solving the problem (29) using (31);
8 Data Allocation:
9 foreach  $r \in \mathcal{R}$  do
10  Get the data allocation strategy  $\lambda_r^d(t)$  by solving the
   problem (28) using algorithm 1;
11 Reducer Selection:
12 Select the reducer to which aggregates the intermediate data
   from Map phase by using (33) (i.e.,  $x_d(t)$  is obtained).
13 Update the queues  $M_d(t), Y_d(t), R_d(t), Z_d(t)$  according to
   queue dynamic equation (17),(18),(19) and (20) respectively.

```

where $d^* = \arg \min_d \{ V \sum_{i \in \mathcal{D}} \Phi_{id}(h_i(t)) + R_d \sum_{i \in \mathcal{D}} h_i(t) \}$.

So far, the three complex problems of data allocation, resource provisioning and reducer selection at time slot t have been solved independently and efficiently. The simple strategies facilitate the online deployment of the algorithm in the real-world systems. Employing the queue updating manner (17),(18),(19) and (20) along time slots, we can design an online algorithm called *MiniBDP* for solving the problems in the long run. The details of the online algorithm are presented in Algorithm 2.

5 PERFORMANCE ANALYSIS

Next, to show its superiority, the performance of the Algorithm ?? in terms of cost optimality, queueing delay bound, and the worst delay of data processing is theoretically analyzed.

Theorem 5.1. (Cost Optimality) Suppose the data generation rate $a_r(t), \forall r \in \mathcal{R}$ is identical an independently distributed over time slots, for any control parameter $V > 0$, the algorithm can achieve a time average cost related with the optimal one as follows.

$$\limsup_{T \rightarrow \infty} \frac{1}{T} \cdot \sum_{t=0}^{T-1} \mathbb{E}\{C(t)\} \leq C^* + \frac{B}{V}, \tag{34}$$

where C^* is the infimum of the time average cost when choosing the optimal control action, representing the theoretically optimal solution, B is the same as defined in (26).

Proof: Please see the Appendix A in the supplemental file.

This theorem exhibits that the gap between the time average cost obtained by the algorithm proposed in this paper and the optimal cost obtained offline is with $\mathcal{O}(1/V)$. In particular, by choosing the control variable V , the time-average cost C is arbitrarily close to the optimal cost C^* .

Theorem 5.2. (Queues Bound) Assume ϵ_d satisfies $\epsilon_d + \sigma_d < \sum_{k \in \mathcal{K}} N_d^{k, \max} v_k$. Let $M_d^{\max}, Y_d^{\max}, R_d^{\max}$ and Z_d^{\max} be the upper bound of queue $M_d(t), Y_d(t), R_d(t)$ and $Z_d(t)$ respectively, we have:

$$Y_d^{\max} = Z_d^{\max} = 2 \frac{V p_d^{\max}}{v_{\min}} + \epsilon_d + \sigma_d, \tag{35}$$

$$(m_d^k(t), n_d^k(t)) = \begin{cases} (0, 0), & \text{if } M_d(t) + Y_d(t) \leq \frac{V p_d^k(t)}{v_k} \cap R_d(t) + Z_d(t) \leq \frac{V p_d^k(t)}{v_k} \\ (N_d^{k, \max}, 0), & \text{if } R_d(t) + Z_d(t) \leq M_d(t) + Y_d(t) \cap M_d(t) + Y_d(t) \geq \frac{V p_d^k(t)}{v_k} \\ (0, N_d^{k, \max}), & \text{if } R_d(t) + Z_d(t) > M_d(t) + Y_d(t) \cap R_d(t) + Z_d(t) \geq \frac{V p_d^k(t)}{v_k} \end{cases} \quad (31)$$

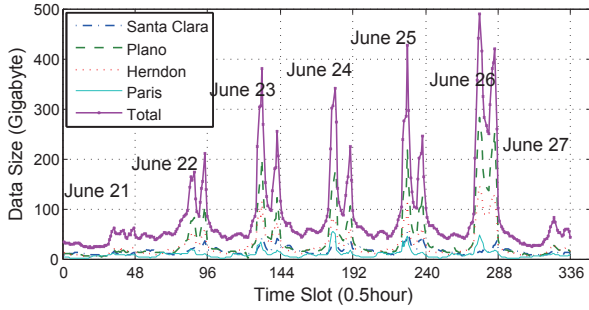


Fig. 2: The data size generation pattern of Worldcup98 web site between June 21-June 27, 1998

and

$$M_d^{\max} = R_d^{\max} = 2 \frac{V p_d^{\max}}{v_{\min}} + \sum_{r \in \mathcal{R}} A_r^{\max} + \mu \beta_{\max} \sum_{d \in \mathcal{D}} \sum_{k \in \mathcal{K}} N_d^{k, \max} v_k, \quad (36)$$

where p_d^{\max} is the max price for each type of VM over time slots, v_{\min} is the minimal capacity among all kinds of VMs, and β_{\max} is the max value of β_r .

Proof: Please see the Appendix B in the supplemental file.

This theorem shows that the queue backlog is with $\mathcal{O}(V)$. It means that, to keep the queue backlog stable, we should choose a small V . Notice that a small V will cause a larger cost as shown in (34), the time-averaged cost and system stability has an $[\mathcal{O}(1/V), \mathcal{O}(V)]$ trade-off. In reality, given the acceptable cost we can choose a suitable V to maximize the system stability, and vice versa.

Theorem 5.3. (Worst Case Delay) Assume that the system runs in the First-in-First-Out manner, the worst delay of the data processing in queue d is bounded by the l defined below:

$$l = \lceil (M_d^{\max} + Y_d^{\max})/\epsilon_d + (R_d^{\max} + Z_d^{\max})/\sigma_d \rceil, \quad (37)$$

where $\lceil x \rceil$ denotes the minimal integer among those greater or equal to x and $M_d^{\max}, Y_d^{\max}, R_d^{\max}, Z_d^{\max}$ are defined in (35) and (36).

Proof: Please see the Appendix C in the supplemental file.

Theorem 5.3 implies that the data arriving at any time slot t can be processed within l time slots using MapReduce framework, which demonstrate our algorithm is able to guarantee the QoS (Quality of Service) for DSP. In addition, given the system parameters, by choosing suitable ϵ_d and σ_d , the QoS for the DSP can be tuned. Also, with different setting of ϵ_d and σ_d for $d \in \mathcal{D}$, we can achieve heterogeneous QoS for different datacenters.

6 EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness of *MiniBDP* using a discrete-event simulator and the real-world traces dataset from world cup 1998 website.

TABLE 2: Average Electricity Price in Different Datacenters

City	Cities in USA	Amsterdam	Dublin	Frankfurt	London
Pirce (\$)	11	28.36	27.81	21.99	28.89

6.1 Dataset Description

As analyzing the huge amount of record data of large-web site (e.g., Youtube, Facebook etc.) is increasingly important for its market decisions, we take user log analytics as an example in the experiment. Unfortunately, the trace logs of the well-known large-scale web sites (e.g., Facebook, LinkedIn) are not open access, we use the WorldCup98 web traces dataset [33] instead to evaluate our algorithm. We believe it does not affect the evaluation result due to its characteristics of dynamism and inscrutability. This dataset records the information of all the requests in the 1998 World Cup Web site between April 30, 1998 and July 26, 1998, which is from 30 servers distributed across four locations used in the web site system (i.e., servers in Paris, France; 10 servers in Herndon, Virginia; 10 servers in Plano, Texas; and 6 servers in Santa Clara, California). Each trace record includes detailed information such as the request time, request client, request object and the server that handled the request etc. We extract one week's data between June 21 to June 27, 1998 from it for experiment. In particular, to simulate the large-scale web site, we augment the amount of original request with 1000x. By aggregating the requests every 30 minutes and setting each record contents to 100bytes, we get the corresponding data volume shown in Fig.2.

6.2 Experiment Setting

In the experiment, we simulate a DSP with 4 datasources which correspond to the servers in four geographic locations (i.e., Santa Clara, Plano, Herndon and Paris) that serves for the WorldCup 1998 web site and a cloud with 12 datacenters in 12 locations corresponding to those of Amazon EC2 in Europe and America (i.e., Ashburn, Dallas, Los Angeles, Miami, Newark, Palo Alto, Seattle, Saint. Louis, Amsterdam, Dublin, Frankfurt and London) [34]. Five types of VM instances (i.e., c3.large, c3.xlarge, c3.2xlarge, c3.4xlarge, c3.8xlarge) provided by EC2 are considered in this paper. Geographic distances between datacenters and datasources are obtained by the online tool in [35], which can be seen in Fig.4(b).

In this paragraph, some different settings are suggested to be itemized one by one. Link delays are set based on Round Trip Time (RTT) among the datasources and data centers, according to their geographic distance (e.g., $RTT \text{ (ms)} = 0.02 \times \text{Distance (km)} + 5$) [36]. The prices of the VMs instance ($p_d^k(t)$) and storage ($s_d(t)$) follow the prices of Amazon EC2 Spot Instance and S3 from the web sites respectively [37], [38]. To stimulate consumption, we assume that the more VMs a customer buys from the CSPs, the cheaper the unit price is. To simulate the VM price change over different datacenters, we set the average electricity price of the city the datacenter located as the price factor. Table 1 shows the electricity price in each city, where 'cities in USA'

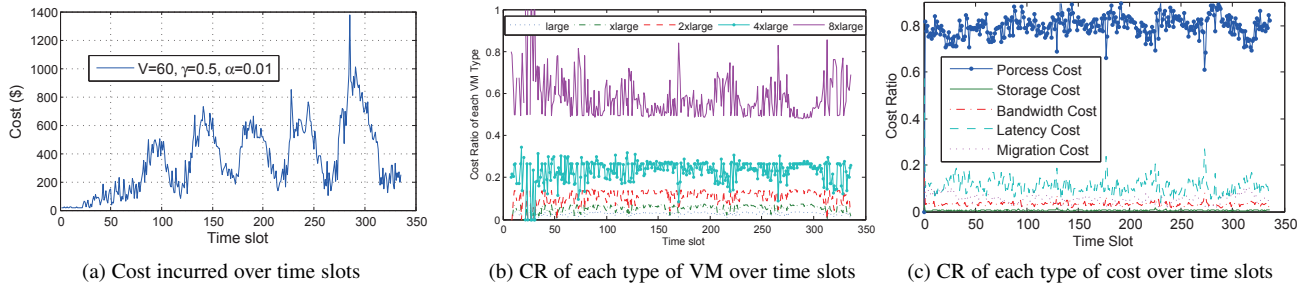
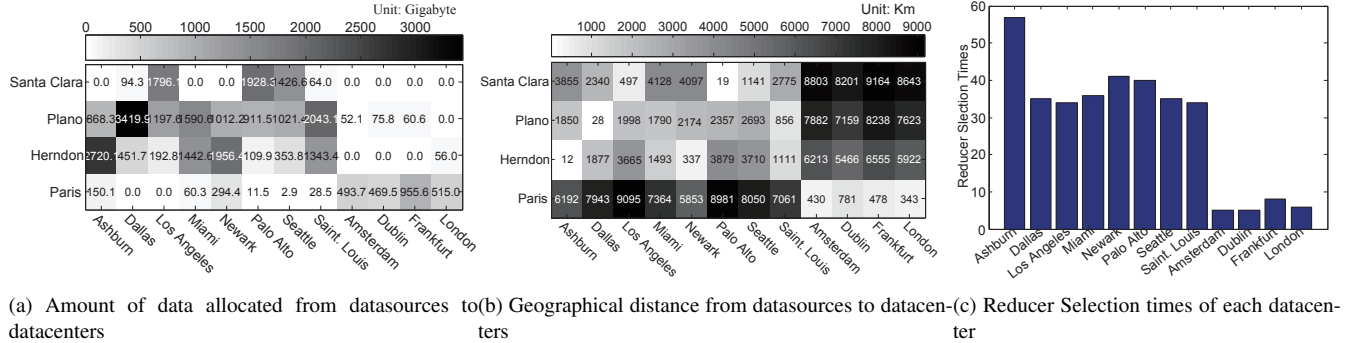


Fig. 3: Cost metric over time slots

Fig. 4: Statistic metric of decision λ and x

include Ashburn, Dallas, Los Angeles, Miami, Newark, Palo Alto, Seattle, Saint. Louis. As suggested by [7], the unit charge of uploading data to datacenters via link $\langle r, d \rangle$ is within $[0.10, 0.25] \text{ \$/GB}$ and the migration cost function is set as a linear function on the amount of data migrated, with a price of inter-datacenter bandwidth in the range of $[0.10, 0.25] \text{ \$/GB}$. We set the uplink bandwidth capacity U_r^d is with uniform distribution within $[30, 50] \text{ M/s}$. For the involvement of historical data, we reuse the past 2 time slots' intermediate data at current time slot (i.e., $\beta_{t-1} > \beta_{t-2} > \beta_{t-3} = \dots = 0$). Unless indicated otherwise, the default experiment setting is adopted as follows: $V = 60, \gamma = 0.5, \alpha = 0.01, \epsilon_d = 1, \sigma_d = \gamma \times \epsilon_d$.

Performance Metrics: In the experiments, two metrics cost and queue size are mainly considered. Cost measures the economic aspects of the system while queue size describes the stability of the system. To facilitate the comparison, Cost Ratio (CR), which measures the cost proportion of a single case among the total cost obtained by all cases, is used in the experiments. It can be calculated by using $CR_{cur} = C_{cur} / \sum_{i=1}^N C_i$, where C_i denotes the cost incurred by the i -th case and N is the case count.

6.3 Performance under Fixed Setting

In this section, we conducted a group of experiments under fixed parameters (the values set for these parameters can be seen in subsection 6.2) to evaluate the effectiveness of *MiniBDP*. Fig.3(a) presents the total cost of the system incurred over time slots. It can be observed that the cost fluctuates synchronously with the data generation pattern as shown in Fig.2, which means that *MiniBDP* is able to adaptively lease and adjust VMs resources to meet dynamic data processing demands even in a flash-crowded style without forecasting the future workload information. This

is substantially different from those existing methods that need prediction phase in algorithm design. Fig.3(b) illustrates the CR comparisons among different VM types, from which we find that the larger the VM capacity is, the more the number of VM with the corresponding type will be rented. This is probably because we design the pricing strategy with the principle that the more capacity of the VM is, the lower the unit price of the VM is. Thus, the algorithm prefers to rent the VM instance with large capacity (e.g., c3.8xlarge) to process the data. The cost components (i.e., processing cost, storage cost, bandwidth cost, latency cost, and migration cost) at each time slot are also compared in Fig.3(c), which shows that processing cost occupies the major part of the total cost and the other types of cost are relatively low. This reveals that the algorithm is able to select the suitable datacenter for data processing while reducing the extra cost.

Furthermore, to exploit the inner property of the algorithm, details of decision on data allocation and reducer selection are presented. Fig.4(a) shows the data allocation matrix map from datasources to datacenters and Fig.4(b) illustrates the corresponding geographic distance map. As can be observed, the algorithm shows data locality property since data are preferred to be moved to the near datacenter for processing. The remoter the datacenter is, the less the data are moved to. In particular, there is hardly any data should be moved from Paris to the datacenters in North America (i.e., Ashburn, Dallas, Los Angeles, Miami, Newark, Palo Alto, Seattle, Saint. Louis) even the price in the Europe is significantly higher than that in the North America. This means that our algorithm is capable of avoiding long latency cost to guarantee the data processing delay. Fig.4(c) depicts the times of reducer selection of each datacenter, which also shows that majority of the Reduce operations are done within the datacenters located in the North America. With this strategy, the migration

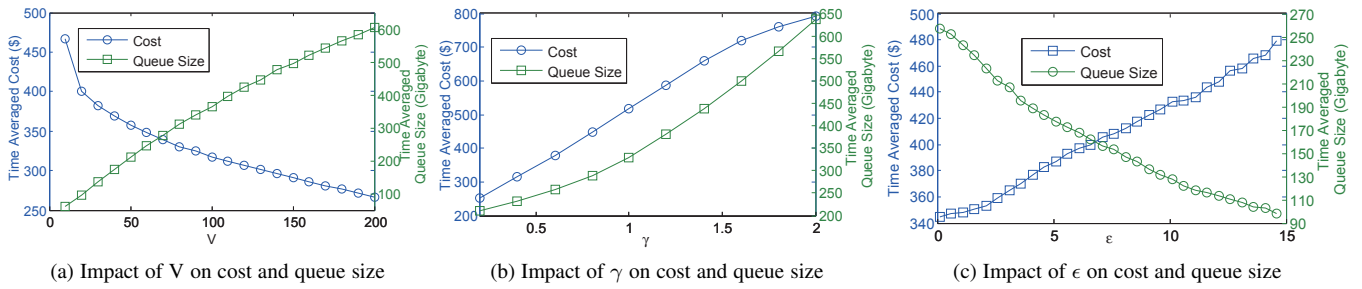


Fig. 5: Impact of important parameters on cost and queue size

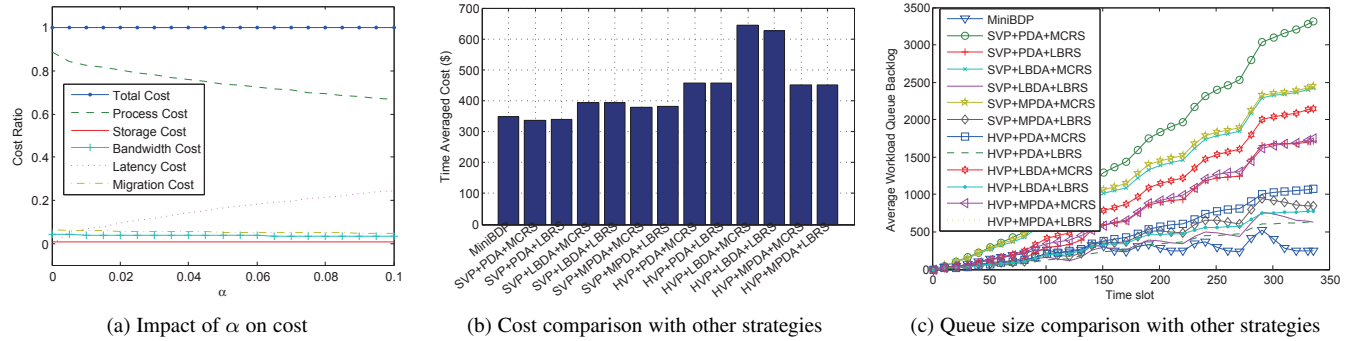


Fig. 6: Comparison with other strategies

cost is reduced since migrating the data from 4 datacenters located in the Europe to 8 datacenters in the North America is more economic than in the opposite route.

6.4 Impact of Parameters

In this section, we investigate the impact of important parameters (i.e., $V, \gamma, \epsilon, \alpha$) on the algorithm.

(1) As presented previously, V is a very important parameter to the model, which controls the trade-off between cost and queue size. Fig.5(a) depicts the cost and workload queue size (i.e., $M_d(t) + R_d(t)$) curve with the variation of V , from which we can observe that the time-averaged cost incurred by the system declines with the increase of V and converges to a minimum value with the increase of V . This provides a guidance for us to reduce the total cost when implementing a real-world system. However, the workload queue size increases with the increase of V . It may lead to the increase of data processing latency since the longer the queue is, the more time it needs to wait for processing. Furthermore, this experimental result is also consistent with the theoretical analysis result of the algorithm in **Theorem 5.1** and **Theorem 5.2**. With this property, given a cost budget, we can choose a suitable V for the system.

(2) The ratio between intermediate data size produced at Map phase and the raw data size imported into Map (i.e., γ) is studied. As can be seen from Fig. 5(b), both cost and queue size grow with the increase of γ . Intuitively, the larger the γ is, the more the intermediate data will be produced. Hence, more resources are needed to process the intermediate data as well more cost will incur in data transportation and storage. When γ increases to a large degree, the cost improves slightly. This may due to the fact that finite amount of VM resources in each datacenter are set, thus all of the VMs will be rented when the γ increases to a specific degree, resulting in a slight change of cost.

(3) To validate the **Theorem 5.3**, we conduct experiments with the variation of ϵ and σ that can be used to control the worst case queue delay. For simplicity, we set the same ϵ and σ for all the datacenters (i.e., $\epsilon_1 = \dots = \epsilon_D = \epsilon, \sigma_1 = \dots = \sigma_D = \sigma$) and $\sigma = \gamma \times \epsilon$. Fig.5(c) illustrates the experimental results, in which we can observe that the cost increases while the queue size decreases with the increase of ϵ . This can be explained by the fact that, with the growth of ϵ and σ , the worst delay l declines (refer to (37)). Therefore, it needs more VMs to process the data within a short delay, and thus resulting in a cost increase. Similarly, the queue length needs to be shortened to guarantee the data processing delay when ϵ and σ increase. Hence, from this point of view, the theoretical result of **Theorem 5.3** is also validated.

(4) The impact of α is also studied by comparing the cost components fluctuation with the change of α . As exhibited in Fig.6(a), the cost of bandwidth and migration cost keep a low level, which means *MiniBDP* is able to avoid unnecessary data moving across datacenters. The graph also shows the costs of storage, bandwidth and migration vary slightly, while the processing cost decreases and the latency cost increases obviously with the increase of α . This may because the costs of storage, bandwidth and migration are mainly determined by the amount of data, and processing cost decreases since latency cost increases with α under a fixed V . Therefore, for a larger latency factor α , we should set a smaller V to guarantee the processing delay.

6.5 Comparisons

In this section, we compare *MiniBDP* with other alternatives, each of which is the combination of a data allocation strategy, VM provisioning strategy and reducer selection strategy.

For the data allocation policies, three representative strategies are considered. 1) Proximity-aware Data Allocation (PDA), in which dynamically generated data from each datasource are

always allocated to the geographically nearest datacenter. It produces minimal latency and is suitable for the scenario that latency delay is prior to other factors. 2) Load-balancing Data Allocation (LBDA), in which the data from each datasource are always dispatched to the datacenter with the lowest Map workload. Obviously, this strategy is capable of keeping workload balanced among datacenters. 3) Minimal Price Data Allocation (MPDA), in which the data from each datasource are allocated to the most economic datacenter, so as to achieve the lowest cost.

For the VM provisioning policies, two typical strategies are considered. 1) Heuristic VM Provisioning (HVP), in which the VMs needed at current time are estimated based on the workload at previous time. To cope with the fluctuation of workload, extra 50 percent VMs are added to those need at previous time to form the final decision. 2) Stable VM Provisioning (SVP), in which the VM count of each type in each datacenter is set to a fixed value. For ease of comparison, we configure the fixed value as the average VM of each type achieved by *MiniBDP*. Thus, the amount of VMs consumed by *SVP* is equal to that of *MiniBDP* within time period T .

For the reducer selection strategies, we consider two approaches as follows. 1) Minimal Migration Cost Reducer Selection (MCRS), this takes the migration cost priority to select the reducer. 2) Load Balance Reducer Selection (LBRS), which selects the datacenter with the smallest workload of Reduce as the reducer.

Therefore, combining with the aforementioned strategies, we have following approaches: *MiniBDP*, *SVP+PDA+MCRS*, *SVP+PDA+LBRS*, *SVP+LBDA+MCRS*, *SVP+LBDA+LBRS*, *SVP+MPDA+MCRS*, *SVP+MPDA+LBRS*, *HVP+PDA+MCRS*, *HVP+PDA+LBRS*, *HVP+LBDA+MCRS*, *HVP+LBDA+LBRS*, *HVP+MPDA+MCRS*, *HVP+MPDA+LBRS*.

Fig.6(b) presents comparison of the time-averaged cost incurred by different strategies. From this graph, we have following observations: (1) Our algorithm *MiniBDP* outperforms the majority of others except *SVP+PDA+MCRS* and *SVP+PDA+LBRS* in terms of cost. This is because these two strategies only move the data to the geographically nearest datacenter, resulting in lowest bandwidth cost and latency cost. Notwithstanding, this strategy is not really effective since their corresponding queue size increases with the increase of time slots (as shown in Fig.6(c)), which means that the system can not keep stable in the long run. As analyzed in section 6.3, *MiniBDP* also has the property of data locality. Hence, compared with these two strategies, *MiniBDP* makes an better trade-off between data locality and system stability. (2) *HVP+LBDA+MCRS* and *HVP+LBDA+LBRS* incur the highest cost. We believe it may due to their load-balanced data allocation strategy in which moving large amount of data from USA to Paris without considering the high bandwidth cost and latency cost on cross-ocean links. From Fig.6(c), it can be seen that only *MiniBDP* is stable in queue size even after running a long time. Whereas, the queue size of other strategies exhibit to be increasing with the time axis, which will certainly lead to the fail of system. Note that *SVP* provisions the same number of VM with *MiniBDP* while causing higher cost and system instability, it can be concluded that *MiniBDP* is capable of optimizing the three decisions to reduce the total cost. As mentioned previously, the VM of *HVP* is provisioned by adding extra 50 percent VMs to those needed at previous time slots. However, those using this VM provisioning strategy in fact do not work well since the queue size of the system is not stable.

Combining Fig.6(b) and Fig.6(c), it seems some simple strategies (e.g., *SVP+LBDA+LBRS*, *SVP+MPDA+LBRS*,

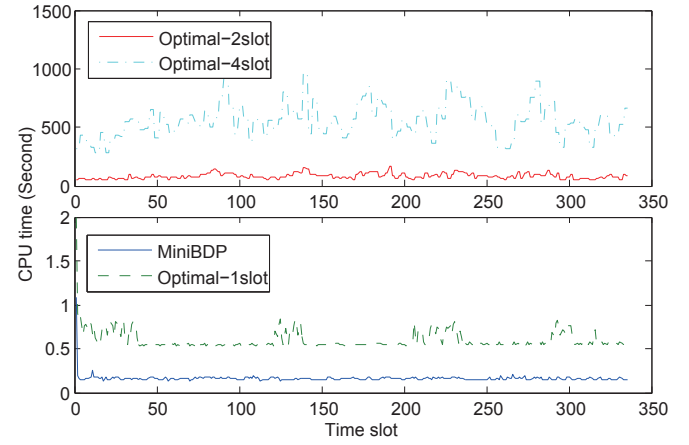
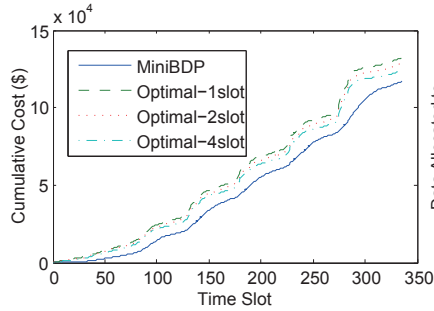


Fig. 7: Solving time Comparison with optimal

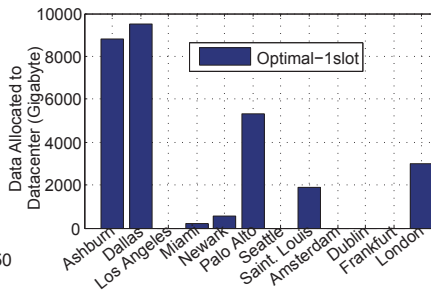
HVP+PDA+LBRS) are close to *MiniBDP*. In fact, these strategies do not perform very well for a long period since the queue sizes show increasing trends. Furthermore, since the number of VMs in strategy *SVP* is obtained based on the result of *MiniBDP*, *SVP* is actually unreachable in reality.

In addition, we also compare *MiniBDP* with the offline optimal ones. Since the original problem contains 60480 variables (at one time slot, \mathbf{m} , \mathbf{n} are with 60 variables, \mathbf{x} is with 12 variables, λ is with 48 variables, thus for 336 time slots, there are 180×336 variables), existing optimization toolboxes (e.g., GLPK, CPLEX, LPSOLVE, etc.) show failed to solve such an extreme large-scale integer nonlinear optimization on a Spur server with Intel(R) Xeon(R) CPU (1.90GHz), 128GB RAM, 64bits Windows operation system. Thus, we divide the long time slots into several parts with an interval and solve each of them instead using BMIBNB with lower solver CPLEX, upper solver FMINCON in Matlab. In fact, these alternatives are with known data arrival and achieve suboptimal offline solutions. Furthermore, the tolerable delays are also fixedly set as *interval* time slots since the data must be processed within *interval* time slot. In this experiment, different *intervals* are set to study its performance. Fig.8 depicts the cumulative cost comparison along time slots (Optimal- x slot means that the interval is x), which shows that *MiniBDP* performs better than the cases *interval* = 1, *interval* = 2 and *interval* = 4 and the larger the interval is, the lower the cost is incurred. We believe this is because: 1) data processing must be completed within 1, 2, 4 time slots for *Optimal-1slot*, *Optimal-2slot* and *Optimal-4slot*, respectively and, 2) a smaller interval needs more VM resource to complete data processing due to its short delay. Whereas, *MiniBDP* has a soft delay control mechanism by setting the ϵ and σ , the cost will be reduced by setting a long tolerable delay. The time used for solving the problem is compared in Fig.7, which shows that it requires only about 0.15 seconds for *MiniBDP* to make decision since it is based on some simple strategies, thus it can run in an online fashion. However, the CPU time consumed by *Optimal-2slot* (average 83.9 seconds), *Optimal-4slot* (average 523.8 seconds) is significantly higher than that of *MiniBDP* because it needs extensive searching time to find the global optimal. Note that *Optimal-1slot* runs relatively fast (only needs 0.4 seconds), this may attribute to the fact that the problem comes into a mixed integer linear optimization that can be efficiently solved by existing tools (e.g., LPSOLVE) when *interval* = 1 since the migration data produced at former time

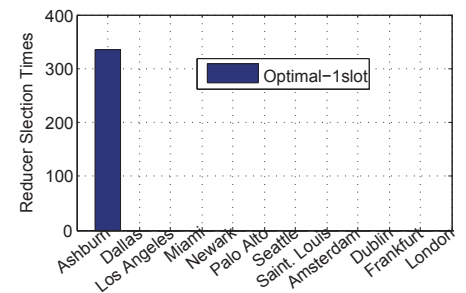


(a) Cumulative cost comparison

Fig. 8: Cost Comparison with Optimal



(a) Data allocation result of Optimal-1slot



(b) Reducer selection result of Optimal-1slot

Fig. 9: Solution of Optimal-1slot

slot is known (i.e., $h_i(t)$ is known, no $x_d(t) * m_d^k(t)$ term exists). We also find that the solving time increases exponentially with the *interval*, e.g., we run *Optimal-2slot* for nearly 8.5 hours and *Optimal-4slot* for more than two days, thus to optimize the original problem in an online manner is almost infeasible. Finally, as can be seen in Fig.9, the solution of *Optimal-1slot* is not able to keep the load balancing among datacenters since it mainly allocates data to only five datacenters and runs reducer in only 1 datacenters. So do the the solutions of *Optimal-2slot* and *Optimal-2slot*, we omit them here due to page limit. Therefore, *MiniBDP* with simple strategies shows many advantages over offline optimal solution and a real-world implementation prospect.

7 CONCLUSION AND FUTURE WORK

With high velocity and high volume of big data generated from geographically dispersed sources, big data processing across geographically distributed datacenters is becoming an attractive and cost effective strategy for many big data companies and organizations. In this paper, a methodical framework for effective data movement, resource provisioning and reducer selection with the goal of cost minimization is developed. We balance five types of cost: bandwidth cost, storage cost, computing cost, migration cost, and latency cost, between the two MapReduce phases across datacenters. This complex cost optimization problem is formulated into a joint stochastic integer nonlinear optimization problem by minimizing the five cost factors simultaneously. By employing Lyapunov technique, we transform the original problem into three independent subproblems that can be solved by designing an efficient online algorithm *MiniBDP* to minimize the long-term time-average operation cost. We conduct theoretical analysis to demonstrate the effectiveness of *MiniBDP* in terms of cost optimum and worst case delay. We perform experimental evaluation using real-world trace dataset to validate the theoretical result and the superiority of *MiniBDP* by compared it with existing typical approaches and offline methods.

The proposed approach is predicted to be with widespread application prospects in those globally-serving companies since analyzing the geographically dispersed datasets is an efficient way to support their marketing decision. As the subproblems in the algorithm *MiniBDP* are with analytical or efficient solutions that guarantee the algorithm running in an online manner, the proposed approach can be easily implemented in the real system to reduce the operation cost. In the future work, we will focus on following aspects: 1) Extending the original model to support other types of jobs. Note the approach is designed mainly for data warehouse

type of job such as statistic analysis and SQL query and some other kinds data processing (e.g., astronomic image processing), the graph type of jobs and the jobs with iteration property are not originally supported in the paper due to the streaming property of the queue designs. However, the proposed approach can be extended to adapt to these cases with minor extensions. E.g., if we change the original model by adding a self-circulation data flow at the data allocation stage and designing a coordinator to transfer the reduce result to mappers, the iterative jobs can be supported. 2) Taking into consideration the factor of data replication in the model. Data replication is well-known as an effective solution for high availability and high fault tolerance. Given that our goal is cost minimization, introducing data replication will add additional cost of replicating data across datacenters. Thus, this factor is not considered in our current cost minimization algorithm and we left it to be one of our ongoing research efforts. 3) In addition, we will concentrate on deploying the proposed algorithm in the real systems such as Amazon EC2 to further validate its effectiveness.

ACKNOWLEDGEMENT

This research is supported by the National Natural Science Foundation of China under grant 61572511, the Natural Science Foundation of Hunan Province under grant 2015JJ3023 as well as the Overseas, Hongkong&Macau Scholars Collaborated Research Fund of China under grant 11428101 as well as the Scientific Research Project of National University of Defense Technology under grants ZK16-03-57 and ZK16-03-09.

REFERENCES

- [1] "Square kilometre array," <http://www.skatelescope.org/>.
- [2] A. Vulimiri, C. Curino, B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proceedings of the USENIX NSDI'15*, 2015.
- [3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the USENIX HotCloud'10*, 2010.
- [5] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational solutions to large-scale data management and analysis," *Nature Reviews Genetics*, vol. 11, no. 9, pp. 647–657, 2010.
- [6] M. Cardoso, C. Wang, A. Nangia et al., "Exploring mapreduce efficiency with highly-distributed data," in *Proceedings of the second international workshop on MapReduce and its applications*, 2011.
- [7] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. M. Lau, "Moving big data to the cloud: An online cost-minimizing approach," *IEEE Journal on Selected Areas in Communications*, vol. 31, pp. 2710–2721, 2013.

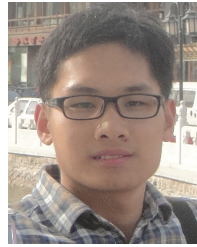
- [8] W. Yang, X. Liu, L. Zhang, and L. T. Yang, "Big data real-time processing based on storm," in *Proceedings of the IEEE TrustCom'13*, 2013.
- [9] Y. Zhang, S. Chen, Q. Wang, and G. Yu, "i2mapreduce: Incremental mapreduce for mining evolving big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, pp. 1906–1919, 2015.
- [10] D. Lee, J. S. Kim, and S. Maeng, "Large-scale incremental processing with mapreduce," *Future Generation Computer Systems*, vol. 36, no. 7, pp. 66–79, 2014.
- [11] B. Heintz, A. Chandra, R. K. Sitaraman, and J. Weissman, "End-to-end optimization for geo-distributed mapreduce," *IEEE Transactions on Cloud Computing*, 2014.
- [12] C. Jayalath, J. Stephen, and P. Eugster, "From the cloud to the atmosphere: Running mapreduce across data centers," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 74–87, 2014.
- [13] P. Li, S. Guo, S. Yu, and W. Zhuang, "Cross-cloud mapreduce for big data," *IEEE Transactions on Cloud Computing*, 2015, doi:10.1109/TCC.2015.2474385.
- [14] A. Sfrant and F. Pop, "Asymptotic scheduling for many task computing in big data platforms," *Information Sciences*, vol. 319, pp. 71–91, 2015.
- [15] L. Zhang, Z. Li, C. Wu, and M. Chen, "Online algorithms for uploading deferrable big data to the cloud," in *Proceedings of the IEEE INFOCOM*, 2014, pp. 2022–2030.
- [16] Q. Zhang, L. Liu, A. Singhand *et al.*, "Improving hadoop service provisioning in a geographically distributed cloud," in *Proceedings of IEEE Cloud'14*, 2014.
- [17] A. Vulimiri, C. Curino, P. B. Godfrey, K. Karanasos, and G. Varghese, "Wanalytics: Analytics for a geo-distributed data-intensive world," in *Proceedings of the CIDR'15*, 2015.
- [18] K. Kloudas, M. Mamede, N. Pregoica, and R. Rodrigues, "Pixida: Optimizing data parallel jobs in wide-area data analytics," *Proceedings of the VLDB Endowment*, vol. 9, no. 2, pp. 72–83, 2015.
- [19] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proceedings of the ACM SIGCOMM'15*, 2015.
- [20] "Facebook's prism project," <http://www.wired.com/wiredenterprise/2012/08/facebook-prism/>.
- [21] J. C. Corbett, J. Dean, M. Epstein *et al.*, "Spanner: Google's globally-distributed database," in *Proceedings of the OSDI'12*, 2012.
- [22] "Connecting geographically dispersed datacenters," *HP*, 2015.
- [23] "Interconnecting geographically dispersed data centers using vpls-design and system assurance guide," *Cisco Systems, Inc.*, 2009.
- [24] L. Tassioulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.
- [25] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Proceedings of the IEEE NOMS*, 2010, pp. 479–486.
- [26] 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.
- [27] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Power cost reduction in distributed data centers: A two-time-scale approach for delay tolerant workloads," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 200–211, 2014.
- [28] D. Wu, Z. Xue, and J. He, "icloudaccess: Cost-effective streaming of video games from the cloud with low latency," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 8, pp. 1405–1416, 2014.
- [29] W. Xiao, W. Bao, X. Zhu, C. Wang, L. Chen, and L. T. Yang, "Dynamic request redirection and resource provisioning for cloud-based video services under heterogeneous environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 1954–1967, 2016.
- [30] S. Rao, R. Ramakrishnan, A. Silberstein *et al.*, "Sailfish: a framework for large scale data processing," in *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012.
- [31] M. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan and Claypool, 2010.
- [32] M. J. Neely, "Opportunistic scheduling with worst case delay guarantees in single and multi-hop networks," in *Proceedings of the IEEE INFOCOM*, 2011, pp. 1728–1736.
- [33] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE Network*, vol. 14, no. 3, pp. 30–37, 2000.
- [34] "Where amazon's data centers are located," <http://www.datacenterknowledge.com/archives/2008/11/18/where-amazons-data-centers-are-located/>.

[35] "Gpsspg," <http://www.gpsspg.com/distance.htm>.

[36] A. Qureshi, "Power-demand routing in massive geo-distributed systems," Ph.D. dissertation, Massachusetts Institute of Technology, 2010.

[37] "Amazon elastic compute cloud," <http://aws.amazon.com/ec2/>.

[38] "Amazon simple storage service," <http://aws.amazon.com/s3/>.



Wenhua Xiao received his B.S. and M.S. degree from the International School of Software at Wuhan University, China, in 2010 and College of Information and System Management at National University of Defense Technology (NUDT), China, in 2012 respectively. He is currently a Ph.D. student in the College of Information System and Management at NUDT. His research interests include scheduling, resource management in cloud computing and big data.



Weidong Bao received his Ph.D. degree in management science and engineering from the National University of Defense Technology in 1999. He is currently a Professor in the College of Information Systems and Management at National University of Defense Technology, Changsha, China. His recent research interests include cloud computing, information system, and complex network. He has published more than 50 research articles in refereed journals and conference proceedings such as IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems and so on. He was a visiting scholar at Arizona State University from May 2013 to May 2014. He serves on the editorial board of AIMS Big Data and Information Analytics.



Xiaomin Zhu received the the PhD degree in computer science from Fudan University, Shanghai, China, in 2009. In the same year, he received the Shanghai Excellent Graduate Award. He is currently an Associate Professor in the College of Information Systems and Management at National University of Defense Technology, Changsha, China. His research interests include scheduling and resource management in green computing, cluster computing, cloud computing, and multiple satellites. He has published more than 70 research articles in refereed journals and conference proceedings such as IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, Journal of Parallel and Distributed Computing, Journal of Systems and Software, and so on. He was a visiting scholar at York University and Georgia Institute of Technology from June 2014 to September 2014 and September 2015 to September 2016, respectively. He serves on the editorial board of Future Generation Computer Systems and AIMS Big Data and Information Analytics. He is a member of the IEEE, the IEEE Communication Society, and the ACM.



Ling Liu is a Professor in the School of Computer Science at Georgia Institute of Technology. She directs the research programs in Distributed Data Intensive Systems Lab (DiSL). She has published over 300 international journal and conference articles and is a recipient of the best paper award from a number of top venues, including ICDCS 2003, WWW 2004, 2005 Pat Goldberg Memorial Best Paper Award, IEEE Cloud 2012, IEEE ICWS 2013, IEEE/ACM CC-Grid 2015. She is an elected IEEE Fellow. She is the editor in chief of IEEE Transactions on Service Computing, and serves on the editorial board of international journals, including ACM Transactions on Web (TWEB), ACM Transactions on Internet Technology (TOIT).