Online Algorithms for Uploading Deferrable Big Data to The Cloud

Linquan Zhang*, Zongpeng Li*, Chuan Wu†, Minghua Chen‡
*University of Calgary, {linqzhan,zongpeng}@ucalgary.ca
†The University of Hong Kong, cwu@cs.hku.hk
‡ The Chinese University of Hong Kong, minghua@ie.cuhk.edu.hk

***Abstract*—This work studies how to minimize the bandwidth cost for uploading deferral big data to a cloud computing platform, for processing by a MapReduce framework, assuming the Internet service provider (ISP) adopts the MAX contract pricing scheme. We first analyze the single ISP case and then generalize to the MapReduce framework over a cloud platform. In the former, we design a *Heuristic Smoothing* algorithm whose worst-case competitive ratio is proved to fall between $2-1/(D+1)$ and $2(1-1/e)$, where D is the maximum tolerable delay. In the latter, we employ the Heuristic Smoothing algorithm as a building block, and design an efficient distributed randomized online algorithm, achieving a constant expected competitive ratio. The Heuristic Smoothing algorithm is shown to outperform the best known algorithm in the literature through both theoretical analysis and empirical studies. The efficacy of the randomized online algorithm is also verified through simulation studies.**

I. Introduction

Cloud computing is emerging as a new computing paradigm that enables prompt and on-demand access to computing resources. As exemplified in Amazon EC2 [1] and Linode [2], cloud providers invest substantially into their data centre infrastructure, providing a virtually unlimited "sea" of CPU, RAM and bandwidth resources to cloud users, often assisted by virtualization technologies. The elastic and on-demand nature of cloud computing assists cloud users to meet their dynamic and fluctuating demands with minimal management overhead, while the cloud ecosystem as a whole achieves *economies of scale* through cost amortization.

Typical computing jobs hosted in the cloud include large scale web applications [3] and big data analytics [4]. In such data-intensive applications, a large volume of information (up to terabytes or even petabytes) is periodically transmitted between the user location and the cloud, through the public Internet. Parallel to utility bill reduction in data centres (computation cost control), bandwidth charge minimization (communication cost control) now represents a major challenge in the cloud computing paradigm [5], [6], [7], where a small fraction of improvement in efficiency translates into millions of dollars in annual savings across the world [8].

Commercial Internet access, particularly the transfer of big data, is nowadays routinely priced by the Internet service providers (ISPs) through a *percentile charge model*, a dramatic departure from the more intuitive total-volume based charge model as in residential utility billing or the flat-rate charge model as in personal Internet and telephone billing [5], [9], [7], [10]. Specifically, in a θ-th percentile charge scheme, the ISP divides the charge period, *e.g.*, 30 days, into small intervals of equal fixed length, *e.g.*, 5 minutes. Statistical logs summarize traffic volumes witnessed in different time intervals, sorted in ascending order. The traffic volume of the θ-th percentile interval is chosen as the charge volume. For example, under the 95th-percentile charge scheme, the cost is proportional to the traffic volume sent in the 8208-th ($95\% \times 30 \times 24 \times 60/5 = 8208$) interval in the sorted list [9], [7], [10]. The *MAX contract model* is simply the 100-th percentile charge scheme. Such percentile charge models are perhaps less surprising when one considers the fact that infrastructure provisioning cost is more closely related to peak instead of average demand.

Due to both its new algorithmic implications and its economic significance in practice, this interesting percentile charge model has soon spawned a series of studies. Most of these endeavours examine cost saving strategies and opportunities through careful traffic scheduling, multihoming (subscribing to multiple ISPs), and inter-ISP traffic shifting. However, they model the cost minimization problem with a critical, although sometimes implicit, assumption that all data generated at the user location have to be uploaded to the cloud *immediately*, without any delay [9], [10]. Consequently, the solution space is restricted to traffic smoothing in the spatial domain only.

Real-world big data applications reveal a different picture, in which a reasonable amount of uploading delay (often specified in service level agreement, or SLA) is tolerable by the cloud user, providing a *golden time window* for traffic smoothing in the temporal domain, which can substantially slash peak traffic volumes and hence communication cost. An example lies in astronomical data from observatories, which are periodically generated at huge volumes but require no urgent attention. Another well-known example is human genome analyses [4], where data are also 'big' but not time-sensitive.

The main challenge of effective temporal domain smoothing lies in the uncertainly in future data arrivals. Therefore a practical cost minimization solution is inherently an *online* algorithm, making periodical optimization decisions based on

This work is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and grants from Hong Kong RGC under the contracts HKU 717812 and HKU 718513.

hitherto input. It is again, surprising, to discover that the online cost minimization for deferrable upload under percentile charging, even when defined over a single link from one source to one receiver only, is still highly non-trivial, exhibiting a rich combinatorial structure, yet never studied before in the literature of either computer networking or theoretical computer science (with an only exception below) [5].

The only study of the online cost minimization problem under percentile charges that we are aware of is a recent work of Golubchik *et al.* [5], which focuses exclusively on the single point-to-point link case. The online algorithm they present, referred to as *Simple Smoothing* here, is extremely simple, and involves evenly smoothing every input across its window of tolerable delay for upload. Nonetheless, this seemingly straightforward algorithm is proven to approach the offline optimum within a small constant under the MAX model. In this work, we first design our own online algorithm for a single link, also adopting the MAX model, in preparation for the MapReduce data processing case. Based on the insight that Simple Smoothing ignores valuable information including the maximum volume recorded so far and the current amount of backlogged data and their deadlines, we tailor a more sophisticated solution, which incorporates a few heuristic smoothing ideas and is hence referred to as *Heuristic Smoothing*. We prove that Heuristic Smoothing always guarantees a competitive ratio no worse than that of Simple Smoothing, under any possible data arrival pattern. Theoretical analysis shows that Heuristic Smoothing can achieve a worst-case competitive ratio between $2 - \frac{1}{D+1}$ and $2(1 - \frac{1}{e})$, where D is the tolerable delay.

We further extend the single link case to a cloud scenario where multiple ISPs are employed to transfer big data dynamically for processing using a MapReduce-like framework. Data are routed from the cloud user to mappers and then reducers, both residing in potentially different data centres of the cloud [6]. We apply Heuristic Smoothing as a plug-in module for designing a distributed and randomized online algorithm with very low computational complexity. The competitive ratio guaranteed by the randomized online algorithm increases from that of Heuristic Smoothing by a small constant factor.

Extensive evaluations are conducted to investigate the performance of the proposed online algorithms. The results show that Heuristic Smoothing performs much better than Immediate Transfer (ITA), a straightforward algorithm that ignores temporal smoothing. Meanwhile Heuristic Smoothing also achieves smaller competitive ratios than Simple Smoothing does. In most cases tested, the observed competitive ratio of Heuristic Smoothing is smaller than 1.5, better than the theoretical upper bound, and relatively close to the offline optimum. Such superior performance is attributed to less abrupt responses to highly volatile traffic demand. Empirical studies for the cloud scenario further verify the efficacy of the randomized cost reduction algorithm, in terms of both scalability and competitive ratio.

In the rest of this paper, we discuss related work in Sec. II, and introduce the system model in Sec. III. Heuristic Smoothing and the randomized algorithm for the cloud scenario are designed and analyzed in Sec. IV and Sec. V, respectively. Evaluation results are in Sec. VI. Sec. VII concludes the paper.

II. Related Work

Similar to deferring data upload to minimize the peak bandwidth demand, there have been studies on scheduling CPU tasks to minimize the maximum CPU speed, that is closely related to the power consumption. Yao *et al.* [11] initially provide an optimal offline algorithm, the YDS algorithm, to optimally minimize power consumption by scaling CPU speed under the assumption that the former is a convex function of the latter. Bansal *et al.* [12] further propose the BKP algorithm with a competitive ratio of e, for minimizing the maximum speed when facing arbitrary inputs with different delay requirements, and arbitrary workload patterns.

Towards new challenges brought by the proliferation of multi-core processors, Albers *et al.* [13] design an online algorithm for multi-processor job scheduling without interprocess job migration. Bingham *et al.* [14] and Angel *et al.* [15] further propose polynomial-time offline optimal algorithms, with migration of jobs considered. Greiner *et al.* [16] generalize a c-competitive online algorithm for a single processor into a randomized cB_α-competitive online algorithm for multiple processors, where B_α is the α-th Bell number. Different from the MAX traffic charge model in this work, they focus on the total volume based energy charges computed by integrating instantaneous power consumption over time.

In recent years, data centre workload scheduling with deadline constraints has been extensively studied in the cloud computing literature. Gupta *et al.* [17] analyze the energy minimization problem in a data center when available deadline information of the workload may be used to defer job execution for reduced energy consumption. Yao *et al.* [18] tackle the power reduction problem with deferrable workloads in date centers using the Lyapunov optimization approach, for approximate time averaged optimization.

A few studies exist on the transfer of big data to the cloud. Cho *et al.* [19] design a static cost-aware planning system for transferring large amounts of data to the cloud provider via both the Internet and courier services. Considering a dynamic transfer scheme where data is produced dynamically, Zhang *et al.* [6] propose two online algorithms to minimize the total transfer cost. Different from this work, they assume mandatory immediate data upload, and adopt a total volume based charge model instead of the percentile charge model.

Goldenberg *et al.* [9] study the multihoming problem under 95-percentile traffic charges. Grothey *et al.* [10] investigate a similar problem through a stochastic dynamic programming approach. They both leverage ISP subscription switching for traffic engineering, so that the charge volume is minimized. However, data traffic in their studies cannot be deferred. Adler *et al.* [20] focus on careful routing of data traffic between two types of ISPs (Average contract, Maximum contract) to pursue the optimal online solution, leading to an online optimization problem similar to the classic ski-rental problem. Golubchik *et al.* [5] study the minimization of transmission cost by exploiting a small tolerable delay when ISPs adopt a 95-percentile or MAX charge model, focusing on a single link only, and proposing the Simple Smoothing algorithm.

III. System model

We consider a cloud user who generates large amounts of data dynamically over time, required for transfer into a cloud or a federation of clouds for processing using a MapReduce-like framework. The mappers and reducers may reside in geographically dispersed data centres. The big data in question can tolerate bounded upload delays specified in their SLA.

A. The MapReduce Framework

MapReduce, initially unveiled by Dean and Ghemawat [21], is a programming model targeting at efficiently processing large datasets in parallel. A typical MapReduce application includes two functions *map* and *reduce*, both written by the users. *Map* processes input key/value pairs, and produces a set of intermediate key/value pairs. The MapReduce library combines all intermediate values associated with the same intermediate key I and then passes them to the *reduce* function. *Reduce* then merges these values associated with the intermediate key I to produce smaller sets of values.

There are four stages in the MapReduce framework: *pushing*, *mapping*, *shuffling*, and *reducing*. The user transfers workloads to the mappers during the pushing stage. The mappers process them during the mapping stage, and deliver the processed data to the reducers during the shuffling stage. Finally the reducers produce the results in the reducing stage. In a distributed system, mapping and reducing stages can happen at different locations. The system will deliver all intermediate data from mappers to reducers during the shuffling stage, and the cloud providers may charge for inter-datacentre traffic during the shuffling stage. Recent studies [22], [23] suggest that the relation between intermediate data size and original data size depends closely on the specific application. For applications such as n-gram models, intermediate data size is much bigger, and the bandwidth cost charged by the cloud provider cannot be neglected. We use β to denote the ratio of original data size to intermediate data size.

B. Cost Minimization for MapReduce Applications

We model a cloud user producing a large volume of data every hour, as exemplified by astronomical observatories [6]. As shown in Fig. 1, the data location is multi-homed with multiple ISPs, for communicating with data centers. Through the infrastructure provided by ISP i, data can be uploaded to a corresponding data centre DCi. Each ISP has its own traffic charge model and pricing function.

After arrival at the data centers, the uploaded data will be processed using a MapReduce-like framework. Intermediate data need to be transferred among data centers in the shuffling stage. Towards a general model, we again assume that multiple ISPs are employed by the cloud to communicate among its distributed data centers, *e.g.*, ISP A for communicating between DC1 and DC2, and ISP B for communicating between DC1 and DC3. If two inter-DC connections are covered by the same ISP, it can be equivalently viewed as two ISPs with identical traffic charge models.

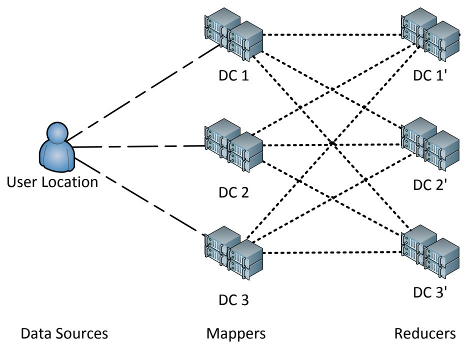

Fig. 1. An illustration of the network for deferrable data upload under the MapReduce framework.

The system runs in a time-slotted fashion. Each time slot is 5 minutes. The charge period is a month (30 days). M and R denote the set of mappers and the set of reducers, respectively. Since each mapper is associated with a unique ISP in the first stage, we employ $m \in M$ to represent the ISP used to connect the user to mapper m. All mappers use the same hash function to map the intermediate keys to reducers [23]. The *upload delay* is defined as the duration between when data are generated to when they are transmitted to the mappers. We focus on uniform delays, *i.e.*, all jobs have the same maximum tolerable delay D, which is reasonable assuming data generated at the same user location are of similar nature and importance. We use W_t to represent each workload released at the user location in time slot t. Let $x_{d,t}^m$ be a decision variable indicating the portion of W_t assigned to mapper m at time slot $t+d$. The cost of ISP m is indicated by $f_m(V_m)$, where V_m is the maximum traffic that goes through ISP m at time slot t.

To ensure all workload is uploaded into the cloud, we have:

$$0 \le x_{d,t}^m \le 1, \forall m \in M. \tag{1}$$

$$\sum_m \sum_{d=0}^{D} x_{d,t}^m = 1, \forall t. \tag{2}$$

Given the maximum tolerable uploading delay D, the traffic V_m^t between the user and mapper m is:

$$V_m^t = \sum_{d=0}^{D} W_{t-d} x_{d,t-d}^m, \forall m \in M. \tag{3}$$

Let V_m be the maximum traffic volume of ISP m, which will be used in the calculation of bandwidth cost. V_m satisfies:

$$V_m - V_m^t \ge 0, \forall t. \tag{4}$$

We assume that ISPs in the first stage, connecting user to mappers, employ the same charging function f_m; and ISPs in the second stage from mappers to reducers use the same charging function $f_{m,r}$. Both charging functions f_m and $f_{m,r}$ are non-decreasing and convex. We further assume that the first stage is non-splittable, *i.e.*, each workload is uploaded through one ISP only.

The user decides to deliver the workload to mapper m in time slot t. Assume it takes a unit time to transmit data via ISPs. Let M_m^{t+1} denote the total data size at mapper m in time slot $t+1$. M_m^{t+1} can be calculated as the summation of all transmitted workloads at time slot t:

$$M_m^{t+1} = \sum_{d=0}^{D} W_{t-d} x_{d,t-d}^m, \forall m \in M.$$

Assume the mappers take 1 time slot to process a received workload. Therefore the mappers will transfer data to the reducer in time slot $t+2$. Let $T_{m,r}^{t+2}$ be the traffic from mapper m to reducer r is in time slot $t+2$:

$$V_{m,r}^{t+2} = \beta M_m^{t+1} y_{m,r}^{t+2}, \forall m \in M, r \in R. \tag{5}$$

The maximum traffic volume of the ISP (m,r), $V_{m,r}$, satisfies:

$$V_{m,r} - V_{m,r}^{t+2} \geq 0, \forall t. \tag{6}$$

Notice that the MapReduce framework partitions the output pairs (key/value) of mappers to reducers using hash functions. All values for the same key are always reduced at the same reducer no matter which mapper it comes from. Furthermore, we assume that data generated in the data locations are uniformly mixed, therefore we have:

$$y_{m,r}^{t+2} = z_r, \forall m \in M, r \in R. \tag{7}$$

This equation also implies that the superscript of $y_{m,r}^{t+2}$ can be ignored. Now we can formulate the overall traffic cost minimization problem for the cloud user, under the MAX contract charge model:

$$\text{minimize} \quad \sum_m f_m(V_m) + \sum_{m,r} f_{m,r}(V_{m,r}) \tag{8}$$

subject to:

$$\begin{aligned}
V_m - V_m^t &\geq 0, & \forall t, m \quad & (8a)\\
V_{m,r} - V_{m,r}^t &\geq 0, & \forall t, m, r \quad & (8b)\\
\sum_{d=0}^{D} x_m^{d,t} &= n_m, & \forall t, m \quad & (8c)\\
\sum_m n_m &= 1, & & (8d)\\
0 \leq x_m^{d,t} \leq 1, & n_m \in \{0,1\}, & \forall m \quad & (8e).
\end{aligned}$$

where V_m^t and $V_{m,r}^t$ are defined in Eqn. (3) and Eqn. (5), respectively. n_m is a binary variable indicating whether ISP m is employed or not.

For ease of reference, notations are summarized l in Tab. I.

IV. The Single ISP Case

We first investigate the basic case that includes one mapper and one reducer only, co-located in the same data center, with no bandwidth cost between the pairs. Given a MAX charge model at the ISP, the algorithm tries to exploit the allowable delay by scheduling the traffic to the best time slot within the allowed time window, for reducing the charge volume. This can be illustrated through a toy example: in $t = 1$, a job ($100MB$, max delay = 9 time slots) is released; in the following time slots, no jobs are released. If the algorithm smooths the traffic across the 10 time slots, the charge volume can be reduced to $10MB/5min$, from $100MB/5min$ if immediate transmission is adopted.

TABLE I
NOTATION

Symbol	Definition
D	the maximum delay from the time data is generated to the time the data location begins to transmit it to the mappers.
M	the set of mappers.
R	the set of reducers. Some mapper and reducer may be in the same location, *i.e.*, $M \cap R \neq \emptyset$.
W_t	the workload released in user location at time slot t.
$x_{d,t}^m$	the portion of the workload W_t that is assigned to mapper m at time slot $t+d$.
β	the ratio of the size of output of a mapper to the size of its input.
$y_{m,r}^t$	the portion of the output of mapper m that is transmitted to reducer r at time slot t.
z_r	the portion of the key space mapped into reducer r.
V_m^t	the total traffic that goes through ISP m at time slot t.
$f_m(y)$	the cost of ISP m for the input $\boldsymbol{y}$.

A. The Primal & Dual Cost Minimization LPs

We can drop the location index (m,r) in this basic scenario of one mapper and one reducer locating in the same data centre. Note that the charging function f is a non-decreasing function of the maximum traffic volume. Minimizing the maximum traffic volume therefore implies minimizing the bandwidth cost. Consequently, the cost minimization problem in our basic single ISP scenario can be formulated into the following (primal) linear program (LP):

$$\text{minimize} \quad V \tag{9}$$

subject to:

$$\begin{aligned}
\sum_{d=0}^{\min\{D,t-1\}} W_{t-d} x_{d,t-d} &\geq V, & \forall t \in \mathcal{T} \quad & (9a)\\
\sum_{d=0}^{D} x_{d,t} &= 1, & \forall t \in \mathcal{T}_D \quad & (9b)\\
x_{d,t} \geq 0, V &\geq 0, & \forall d \in \mathcal{D}, t \in \mathcal{T}_D, \quad & (9c)
\end{aligned}$$

where $\mathcal{T} = [1,T], \mathcal{T}_D = [1, T-D]$, $\mathcal{D} = [0,D]$ and $x_{d,t} = 0, \forall t > T-D, \forall d \in \mathcal{D}$

Introducing dual variable $\boldsymbol{y}$ and $\boldsymbol{z}$ to constraints $(9a)$ and $(9b)$ respectively, we formulate the corresponding dual LP:

$$\text{maximize} \quad \sum_{t=1}^{T-D} z_t \tag{10}$$

subject to:

$$\begin{aligned}
\sum_{t=1}^{T} y_t &\leq 1 & & (10a)\\
z_t - W_t y_{t+d} &\leq 0, & \forall t \in \mathcal{T}_D, d \in \mathcal{D} \quad & (10b)\\
y_t &\geq 0, & \forall t \in \mathcal{T} \quad & (10c)\\
z_t \text{ unconstrainted}, & & \forall t \in \mathcal{T}_D \quad & (10d)
\end{aligned}$$

The input begins with W_1 and ends with W_{T-D}, and $W_{T-D+1} = 0, ..., W_T = 0$ is padded to the tail of the input. We use $\mathcal{P}$ and $\mathcal{D}$ to denote feasible solutions to the primal and dual LPs, respectively.

The optimization in (9) is a standard linear program. For an offline optimal solution, one can simply solve (9) using a standard LP solution algorithm such as the simplex method or the interior-point method.

B. Online algorithms

The simplest online solution in the basic one ISP scenario is the *immediate transfer algorithm* (ITA). Once a new job arrives, ITA transfers it to mappers immediately without any delay. Next we analyze the competitive ratio of ITA, as compared to the offline optimum.

Theorem 1. *ITA is $(D+1)$-competitive.*

Proof: Consider the following input: $(W, 0, 0, 0, 0, ...)$. ITA will process it immediately with bandwidth cost: W. However the offline optimal algorithm will divide the workload into small pieces: $W/(D+1), W/(D+1), ...W/(D+1), 0, 0, 0, ...)$, feasible within the deadline D, with maximum traffic volume $W/(D+1)$.

$$\text{Competitive ratio } \lambda \geq \frac{W}{W/(D+1)} = D + 1$$

We hence obtain a lower bound on the competitive ratio of ITA, $D+1$. Next we prove $D+1$ is also an upper-bound. Without exploiting any delays, ITA provides a feasible solution to the primal problem, which is denoted as $\mathcal{P}_{ITA}$.

$$\mathcal{P}_{ITA} = \max_t W_t$$

Now we design a feasible solution to the dual problem as follows (assume $\tau = \arg max_t W_t$):

$$y_t = \begin{cases} 1/(D+1) & \text{if } t = \tau, ..., \tau + D \\ 0 & otherwise \end{cases}$$

$$z_t = \begin{cases} 1/(D+1)W_t & \text{if } t = \tau \\ 0 & otherwise \end{cases}$$

$$\mathcal{D} = \frac{1}{D+1} W_\tau$$

So the competitive ratio is:

$$\text{Competitive ratio } \lambda = \frac{\mathcal{P}_{ITA}}{OPT} \leq \frac{\mathcal{P}_{ITA}}{\mathcal{D}} = D + 1 \qquad \square$$

Remarks: if $D = 0$, *i.e.* jobs are not deferrable, the offline optimal algorithm degrades into ITA, agreeing with the theorem, which claims ITA is 1-competitive ($D + 1 = 1$).

ITA is apparently not ideal, and may lead to high peak traffic and high bandwidth cost as compared with the offline optimum. Golubchik *et al.* [5] design a cost-aware algorithm that strikes to spread out bandwidth demand by utilizing all possible delays, referred to as the *Simple Smoothing Algorithm*. Upon receiving a new workload, Simple Smoothing evenly divides it into $D+1$ parts, and processes them one by one in the current time slot and the following D time slots, as shown in Algorithm 1.

Algorithm 1 The Simple Smoothing Algorithm [5]

```
1: for $\tau = 1$ to $T - D$ do
2:     for $d = 0$ to $D$ do
3:         $x_{d,\tau} = 1/(D+1)$
4:     end for
5: end for
```

Theorem 2. *[5] The competitive ratio of Simple Smoothing is* $2 - \frac{1}{D+1}$.

Theorem 2 can be proven through weak LP duality, *i.e.*, using a feasible dual as the lower bound of the offline optimal.

Simple Smoothing is very simple, but guarantees a worst case competitive ratio smaller than 2. Nonetheless, there is still room for further improvements, since Simple Smoothing ignores available information such as the hitherto maximum traffic volume transmitted, and the current "pressure" from backlogged traffic and their deadlines. Such an observation motivated our design of the more sophisticated Heuristic Smoothing algorithm for the case $D \geq 1$, as shown in Algorithm 2. Here T is the charge period, τ is the current time slot, and H_d is the total volume of data that have been buffered for d time slots.

Algorithm 2 The Heuristic Smoothing Algorithm

```
1: $V_{max} = 0$
2: $W_\tau = 0, \forall \tau = T - D + 1, ..., T$;
3: $H_d = 0, \forall d = 1, ..., D$;
4: for $\tau = 1$ to $T$ do
5:     $V_\tau = \min\left\{W_\tau + \sum_{d=1}^{D} H_d, \max\{V_{max}, \frac{W_\tau}{D+1} + \frac{\sum_{d=1}^{D} H_d}{D}\}\right\}$
6:     if $V_{max} < V_\tau$ then
7:         $V_{max} = V_\tau$;
8:     end if
9:     Transfer the traffic following Earliest Deadline First (EDF) strategy;
10:    Update $H_d, \forall d = 1, ..., D$;
11: end for
```

Theorem 3. *The competitive ratio of Heuristic Smoothing is lower bounded by* $2(1 - \frac{1}{e})$.

Proof: Consider the following input: $(W, W, ...W, 0, ..., 0)$ whose first $D+1$ time slots are W.

The traffic demand V increases until time slot $D+1$.

$$V_{D+1} = \frac{W}{D+1} + \frac{W}{D+1} + \frac{(D-1)W}{(D+1)D} + ... + \frac{(D-1)^{D-1}W}{(D+1)D^{D-1}}$$
$$= \frac{W}{D+1}(1 + D(1 - (1 - \frac{1}{D})^D))$$

We can find a feasible primal solution which yields the charge volume $\frac{D+1}{2D+1}W$. This primal solution is an upper bound of the offline optimum. Therefore the lower bound of the competitive ratio $\lambda \geq \frac{2D+1}{V_{D+1}(D+1)} = \frac{2D+1}{(D+1)^2}(1 + D(1 - (1 - \frac{1}{D})^D)) \rightarrow 2(1 - \frac{1}{e})$ as $D \rightarrow +\infty$. Notice that

$\frac{2D+1}{(D+1)^2}(1+D(1-(1-\frac{1}{D})^D))$ is a decreasing function for $D \in [1,+\infty)$, we further have $\lambda \geq 2(1-\frac{1}{e})$. $\square$

Theorem 4. *The competitive ratio of Heuristic Smoothing is upper-bounded by* $2-\frac{1}{D+1}$.

Proof: We take the Simple Smoothing algorithm (Algorithm. 2) as a benchmark, and we prove that $\mathcal{P}_{smooth} \geq \mathcal{P}_{heuristic}$, where $\mathcal{P}_{heuristic}$ is the charged volume produced by Algorithm 3.

Algorithm 3 will only increase the traffic demand when $\frac{W_\tau}{D+1}+\sum_{d=1}^{D} H_d/D$ exceeds V_{max}. Therefore, we rearrange H_d to compute the maximum traffic demand. Let

$$\begin{aligned}V_{t+D} = (&\frac{W_{t+D}}{D+1}+\frac{W_{t+D-1}}{D+1}\\ &+\frac{(D-1)W_{t+D-2}}{(D+1)D}+...+\frac{(D-1)^{D-1}W_t}{(D+1)D^{D-1}})\end{aligned}$$

Then $\mathcal{P}_{heuristic} = \max_t V_{t+D}$. Let $\tau = \arg\max_t V_{t+D}$, and we have

$$\begin{aligned}\mathcal{P}_{smooth} &= \max_t \sum_{i=t}^{t+D} \frac{W_t}{D+1}\\ &\geq \sum_{i=\tau}^{\tau+D} \frac{W_\tau}{D+1}\\ &\geq \frac{W_{\tau+D}}{D+1}+\frac{W_{\tau+D-1}}{D+1}+\frac{(D-1)W_{\tau+D-2}}{(D+1)D}\\ &\quad +...+\frac{(D-1)^{D-1}W_\tau}{(D+1)D^{D-1}}\\ &= \mathcal{P}_{heuristic}\end{aligned}$$

Since the simple smoothing algorithm is $2-\frac{1}{D+1}$–competitive, the competitive ratio of Algorithm 3 cannot be worse than $2-\frac{1}{D+1}$. $\square$

From the proof above, we have following corollary.

Corollary 1. *For any given input, the charge volume resulting from Heuristic Smoothing is always equal to or smaller than that of Simple Smoothing.*

Algorithm Complexity. All three online algorithms discussed have moderate time complexity, making them light-weight for practical applications. More specifically, ITA, Simple Smoothing and Heuristic Smoothing have a time complexity of $O(T-D)$, $O((T-D)D)$, and $O(TD)$, respectively.

V. Cloud Scenario

In this section, we apply the algorithms designed for the single ISP case to the cloud scenario, which utilizes a MapReduce-like framework for processing big data. Define $Cost_1 = \sum_m f_m(V_m)$, $Cost_2 = \sum_{m,r} f_{m,r}(V_{m,r})$, and adopt power charge functions by letting $f_m(x) = f_{m,r}(x) = x^\alpha, \alpha > 1$.

A. Algorithm Design

The two-phase MapReduce cost optimization problem is defined in (8), and is a discrete optimization with integer variables. Consequently, an offline solution that solves such an integer program has a high computational complexity, further motivating the design of an efficient online solution.

A native online algorithm selects a fixed mapper and schedules the traffic on the corresponding ISP using the Simple Smoothing Algorithm.

Theorem 5. *The competitive ratio of the native online algorithm is lower bounded by* $|M|^{\alpha-1}$, *where* $|M|$ *is the number of mappers.*

Proof: Consider the input $(W,\cdots,W,0,\cdots,0)$ whose first $D+1$ items are W. We can verify that the charge volume is $\frac{DW}{2D+1}$. The corresponding cost is $(\frac{DW}{2D+1})^\alpha+\sum_r(\beta z_r\frac{DW}{2D+1})^\alpha$.

Next we consider a more intelligent algorithm that assigns the j-th workload to the mapper ($j \mod |M|$). This algorithm acts as the upper bound of the offline optimum. Its charge volume is $\frac{DW}{(2D+1)|M|}$. The corresponding cost is $|M|(\frac{DW}{(2D+1)|M|})^\alpha+|M|\sum_r(\beta z_r\frac{DW}{(2D+1)|M|})^\alpha$. Therefore,

$$\begin{aligned}&\text{Competitive ratio}\\ &\geq \frac{(\frac{DW}{2D+1})^\alpha+\sum_r(\beta z_r\frac{DW}{2D+1})^\alpha}{|M|(\frac{DW}{(2D+1)|M|})^\alpha+|M|\sum_r(\beta z_r\frac{DW}{(2D+1)|M|})^\alpha}\\ &= |M|^{\alpha-1}\end{aligned}$$ $\square$

We next present a distributed randomized online algorithm for (8). For each workload, the user chooses ISPs uniformly at random to transfer the data to a randomly selected mapper. Formally, let WA be the randomized workload assignment allocating each workload to mappers. For each selected ISP, the user runs Heuristic Smoothing to guide one-stage traffic deferral and transmission, as shown in Algorithm 3.

Algorithm 3 Randomized Uploading Scheme

```
1: Generate a randomized workload assignment $WA$ which
   allocates each workload to a randomly selected mapper.
2: For each ISP $m$, apply the single ISP algorithm, e.g.,
   Algorithm 2 to schedule the traffic.
```

We analyze Algorithm 3 by building a connection between the uploading scheme π and the randomized workload assignment WA. We combine π and WA to a new uploading scheme π_{WA}. Let $t_0 = 1 < t_1 \cdots < t_e = T$. During each interval $[t_i, t_{i+1})$, each ISP is employed to transfer at most one workload in the uploading scheme π. If a workload is processed in $[t_i, t_{i+1})$, then it cannot be finished before t_{i+1}. Due to the MAX charge model, the transfer speed for workload w in $[t_i, t_{i+1})$ is a single speed, say $v_{i,w}$. If workload w is not processed in $[t_i, t_{i+1})$, we set $v_{i,w} = 0$. Therefore, for any given i, there are at most $|M|$ values of $v_{i,w} \neq 0$.

Assume there are n workloads, forming a set $\mathbb{W}$. Let $\Omega_m = \{w|\text{all workloads assigned to ISP } m\} \in \mathbb{W}$. In scheme π_{WA}, the user transfers data at speed of $\sum_{w\in\Omega_m} v_{i,w}$ in time interval $[t_i, t_{i+1})$. Let $\phi_n(\Omega_m)$ be the probability that exactly the workloads Ω_m are allocated to ISP m.

$$\phi_n(\Omega_m) = (\frac{1}{|M|})^{|\Omega_m|}(1-\frac{1}{|M|})^{n-|\Omega_m|}$$

We next define function $\Lambda_n(\boldsymbol{x})$ where $\boldsymbol{x} \in \mathbb{R}^n \setminus \{0\}$:

$$\Lambda_n(\boldsymbol{x}) = |M| \sum_{\Omega_m \in \mathbb{W}} \phi_n(\Omega_m)(\sum_{w \in \Omega_m} x_w)^\alpha / \sum_{w=1}^{n} x_w^\alpha$$

Lemma 1. *Given any uploading scheme π and a randomized workload assignment WA, we have a randomized uploading scheme π_{WA}, which satisfies:*

$$\begin{aligned}&\mathbb{E}(Cost_1(\pi_{WA}) + Cost_2(\pi_{WA})) \\ &\quad \le \max_{\boldsymbol{x}} \Lambda_{|M|}(\boldsymbol{x})(Cost_2(\pi) + Cost_1(\pi))\end{aligned}$$

Proof: Since the traffic pattern in ISP $(m,r), \forall r$ is exactly the same as ISP m, we only consider one stage. Let us consider scheme π first. In the first stage, the cost is:

$$Cost_1(\pi) = \sum_{m \in M} \max_{i,w}(v_{i,w}^m)^\alpha \ge \max_i \Sigma^*_{|M|}(v_{i,w}^\alpha)$$

where $v_{i,w}^m$ indicates the transfer speed in ISP m during $[t_i, t_{i+1})$ for workload w. $\Sigma^*_{|M|}(v_{i,w}^\alpha)$ is the sum of the largest $|M|$ values of $v_{i,w}^\alpha$ when given i. The inequality holds because there are at most $|M|$ non-zero speeds for any given duration $[t_i, t_{i+1})$. We next have the cost of the second stage:

$$\begin{aligned}Cost_2(\pi) &= \sum_m \sum_r \max_{i,w}(\beta z_r v_{i,w}^m)^\alpha \\ &= \beta^\alpha \sum_r z_r^\alpha \sum_m \max_{i,w}(v_{i,w}^m)^\alpha \\ &\ge \beta^\alpha \sum_r z_r^\alpha \max_i \Sigma^*_{|M|}(v_{i,w}^\alpha)\end{aligned}$$

The cost of the first stage in π_{WA} is:

$$\begin{aligned}\mathbb{E}(Cost_1(\pi_{WA})) &= \sum_{m \in M} \sum_{\Omega_m^{WA} \in \mathbb{W}} \phi_n(\Omega_m^{WA}) \max_i (\sum_{w \in \Omega_m^{WA}} v_{i,w})^\alpha \\ &= |M| \max_i \sum_{\Omega_m^{WA} \in \mathbb{W}} \phi_n(\Omega_m^{WA})(\sum_{w \in \Omega_m^{WA}} v_{i,w})^\alpha\end{aligned}$$

The second equality above holds because the assignment is uniformly random. Similarly, The cost of the second stage in π_{WA} is:

$$\begin{aligned}&\mathbb{E}(Cost_2(\pi_{WA})) \\ &= |M| \sum_{\Omega_m^{WA} \in \mathcal{P}} \phi_n(\Omega_m^{WA}) \sum_r \max_i (z_r \sum_{w \in \Omega_m^{WA}} \beta v_{i,w})^\alpha \\ &= |M| \beta^\alpha \sum_r z_r^\alpha \max_i \sum_{\Omega_m^{WA} \in \mathbb{W}} \phi_n(\Omega_m^{WA})(\sum_{w \in \Omega_m^{WA}} v_{i,w})^\alpha\end{aligned}$$

Again because for any $[t_i, t_{i+1})$, there are at most $|M|$ values of $v_{i,w} \neq 0$. We have

$$\begin{aligned}&\frac{|M| \sum_{\Omega_m^{WA} \in \mathbb{W}} \phi_n(\Omega_m^{WA})(\sum_{w \in \Omega_m^{WA}} v_{i,w})^\alpha}{\Sigma^*_{|M|}(v_{i,w}^\alpha)} \\ &= \frac{|M| \sum_{\Omega_m^{WA} \in \mathbb{W}} \phi_n(\Omega_m^{WA})(\sum_{w \in \Omega_m^{WA}} v_{i,w})^\alpha}{\sum_{w=1}^n (v_{i,w}^\alpha)} \\ &= \Lambda_n(\boldsymbol{v}) = \Lambda_{|M|}(\boldsymbol{v}')\end{aligned}$$

where $\boldsymbol{v}'$ is an $|M|$-dimensional subvector of $\boldsymbol{v} \in \mathbb{R}^n \setminus \{0\}$, which contains all non-zero transfer speeds in $[t_i, t_{i+1})$.

Therefore, the ratio for the first stage is:

$$\begin{aligned}&\frac{\mathbb{E}(Cost_1(\pi_{WA}))}{Cost_1(\pi)} \\ &\le \frac{|M| \sum_{\Omega_m^{WA} \in \mathbb{W}} \phi(\Omega_m^{WA}) \max_i (\sum_{w \in \Omega_m^{WA}} v_{i,w})^\alpha}{\max_i \Sigma^*_{|M|}(v_{i,w}^\alpha)} \\ &\le \frac{|M| \sum_{\Omega_m^{WA} \in \mathbb{W}} \phi(\Omega_m^{WA})(\sum_{w \in \Omega_m^{WA}} v_{i^*,w})^\alpha}{\Sigma^*_{|M|}(v_{i^*,w}^\alpha)} \\ &\le \max_{\boldsymbol{x}} \Lambda_{|M|}(\boldsymbol{x})\end{aligned}$$

where $i^* = \arg\max_i (\sum_{w \in \Omega_m^{WA}} v_{i,w})^\alpha$. Similarly, the ratio for the second stage is also bounded by $\max_{\boldsymbol{x}} \Lambda_{|M|}(\boldsymbol{x})$, *i.e.*, $\frac{\mathbb{E}(Cost_2(\pi_{WA}))}{Cost_2(\pi)} \le \max_{\boldsymbol{x}} \Lambda_{|M|}(\boldsymbol{x})$. This proves Lemma 1. $\square$

Let $S(\alpha, j)$ be the j-th Stirling number for α elements, defined as the number of partitions of a set of size α into j subsets [24]. Let B_α be the α-th Bell number, defined as the number of partitions of a set of size α [24]. The Bell number is relatively small when α is small: $B_1 = 1, B_2 = 2, B_3 = 5, B_4 = 15$. The definitions also imply:

$$\sum_j^\alpha S(\alpha, j) = B_\alpha$$

The following lemma is proven by Greiner *et al.* [16].

Lemma 2. *[16] $\forall \alpha \in \mathbb{N}$ and $\alpha \le |M|$,* $\max_{\boldsymbol{x}} \Lambda_{|M|}(\boldsymbol{x}) = \sum_{j=1}^\alpha S(\alpha, j) \frac{|M|!}{|M|^j (|M|-j)!}$.

Theorem 6. *Given a λ-competitive algorithm with respect to cost for the single ISP case, then the randomized online algorithm is $\lambda B_{\lceil \alpha \rceil}$-competitive in expectation.*

Proof: Let π^* be the optimal uploading scheme, the corresponding randomized uploading scheme is π^*_{WA}. The algorithm we use is π_{WA}. Since the workloads in π^*_{WA} and π_{WA} are the same, we have:

$$\mathbb{E}(Cost_1(\pi_{WA})) \le \lambda \mathbb{E}(Cost_1(\pi^*_{WA})) \tag{11}$$

since the algorithm is λ-competitive. Similarly,

$$\mathbb{E}(Cost_2(\pi_{WA})) \le \lambda \mathbb{E}(Cost_2(\pi^*_{WA})) \tag{12}$$

since the traffic pattern in ISP $(m,r), \forall r$ is exactly the same as in ISP m.

Lemma 1 implies:

$$\begin{aligned}&\mathbb{E}(Cost_1(\pi^*_{WA}) + Cost_2(\pi^*_{WA})) \\ &\quad \le \max_{\boldsymbol{x}} \Lambda_{|M|}(\boldsymbol{x})(Cost_1(\pi^*) + Cost_2(\pi^*))\end{aligned} \tag{13}$$

Since $\Lambda_{|M|}(\boldsymbol{x})$ is a monotonically increasing function of α, we use $\lceil \alpha \rceil$ as an upper bound of $\alpha > 1$, obtaining a corresponding upper bound of $\Lambda_{|M|}(\boldsymbol{x})$. Combining Eqn. (11) (12) and (13) as well as Lemma 2, we have the following expected cost of the randomized online algorithm:

$$
\begin{aligned}
&\mathbb{E}(Cost_1(\pi_{WA}) + Cost_2(\pi_{WA})) \\
&\quad \le \lambda \mathbb{E}(Cost_1(\pi^*_{WA}) + Cost_2(\pi^*_{WA})) \\
&\quad \le \lambda \max_{\boldsymbol{x}} \Lambda_{|M|}(\boldsymbol{x})(Cost_1(\pi^*) + Cost_2(\pi^*)) \\
&\quad = \lambda \sum_{j=1}^{\lceil\alpha\rceil} S(\lceil\alpha\rceil, j) \frac{|M|!}{|M|^j(|M|-j)!}(Cost_1(\pi^*) + Cost_2(\pi^*)) \\
&\quad \le \lambda \sum_{j=1}^{\lceil\alpha\rceil} S(\lceil\alpha\rceil, j)(Cost_1(\pi^*) + Cost_2(\pi^*)) \\
&\quad \le \lambda B_{\lceil\alpha\rceil} OPT \qquad \square
\end{aligned}
$$

Remark: For a single link, we can employ Heuristic Smoothing, whose competitive ratio is smaller than 2 with respect to maximum traffic volume. Then the competitive ratio of Algorithm 2 is 2^α in cost. Thus Algorithm 3 is $2^\alpha B_\alpha$-competitive in expectation. When $\alpha = 2$, the competitive ratio is 8, a constant regardless of the number of mappers.

VI. Performance Evaluation

We have implemented Simple Smoothing, Heuristic Smoothing, as well as the randomized online algorithm, for performance evaluation through simulation studies. The default input W_t is generated uniformly at random, as shown in Fig. 2, where all data are normalized, *i.e.*, scaled down by $\max_t W_t$. We assume there are 5 mappers at different locations, and 5 reducers at different locations. We choose $\alpha = 2$, thus the charge function $f_m(x) = f_{m,r}(x) = x^2$.

A. The Single ISP Case

First we compare Heuristic Smoothing with Simple Smoothing. The two algorithms are executed under a delay requirement $D = 5$. Fig. 3 illustrates the traffic volume scheduled at each time slot. Compared with Simple Smoothing, Heuristic Smoothing results in a maximum traffic volume this is about 28% smaller. Heuristic Smoothing tries to exploit the available delay to average the traffic and is less sensitive to the fluctuation of traffic demand, as compared with Simple Smoothing. For example, at around $t = 10$, the traffic of Simple Smoothing increases abruptly due to high traffic demand in the input; around $t = 40$, it goes down due to low traffic demand. In comparison, Heuristic Smoothing results in more even traffic distributions around $t = 10$ and $t = 40$.

Next we examine how the tolerable delay affects the performance of the proposed online algorithms. We execute Simple Smoothing, Heuristic Smoothing and ITA against a variety of delays ranging from $D = 0$ to $D = 24$. We also compute the offline optimum as a benchmark. The observed competitive ratios are shown in Fig. 4. The results suggest that both Simple Smoothing and Heuristic Smoothing perform much better than ITA. Heuristic Smoothing also beats Simple Smoothing, by a smaller margin. Heuristic Smoothing approaches the offline optimum rather closely; the observed competitive ratios are always below 1.5 and usually around 1.2, much better than the theoretically proven upper bound in Theorem 4.

Heuristic Smoothing is further evaluated under other random inputs, including Poisson distribution in Fig. 5, Gaussian distribution in Fig. 6 and a specifically designed random input in Fig. 7. All results verify that Heuristic Smoothing works best among the three online cost minimization algorithms.

B. The Cloud Scenario

We implemented the randomized algorithm in Algorithm 3 and the native algorithm in Sec. V-A. They are evaluated under three types of inputs: uniform distribution, Poisson distribution and Gaussian distribution. We compare the costs of the two algorithms using these inputs, as shown in Fig. 8, Fig. 9 and Fig. 10, respectively. We observe that the randomized algorithm achieves much lower cost than the native algorithm, in particular with longer tolerable delays. For example, Fig. 8 shows that the randomized algorithm saves approximately 45% cost as compared with the native algorithm when $D = 5$, and it saves more than 68% when $D = 10$. This suggests that longer tolerable delays provide the randomized algorithm more space of maneuver, leading to more evident cost reduce.

We further investigate the influence of β, the ratio of original data size to the intermediate data size. Results are shown in Fig. 11. When D is small, a large β causes a rather high cost. However when a large D is used, *e.g.*, $D = 40$, even a large β only produces a relatively small cost.

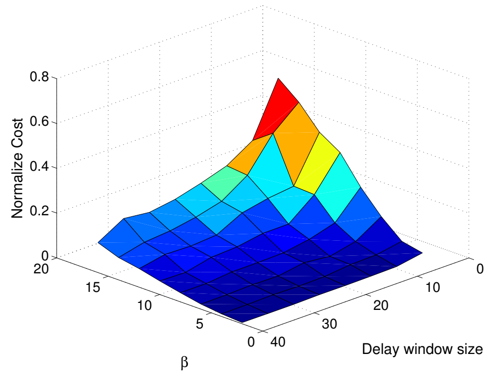

Fig. 11. Relationship between traffic cost and parameters D, β.

VII. Conclusion

ISPs now charge big data applications with a new, interesting percentile based model, leading to new online algorithm design problems for minimizing the traffic cost paid for uploading big data to the cloud. We studied two scenarios for such online algorithm design in this work. A Heuristic Smoothing algorithm is proposed in the single link case, with proven better performance than the best alternative in the literature, and a smaller competitive ratio below 2. A randomized online algorithm is designed for the MapReduce framework, achieving a constant competitive ratio by employing Heuristic Smoothing as a building module. We have focused on MAX charge rules, and leave similar online algorithm design for 95-percentile charge rules as future work.

References

[1] *Amazon Elastic Compute Cloud*, http://aws.amazon.com/ec2/.
[2] *Linode*, https://www.linode.com/speedtest/.
[3] *Amazon EC2 Case-studies*, http://aws.amazon.com/solutions/case-studies.

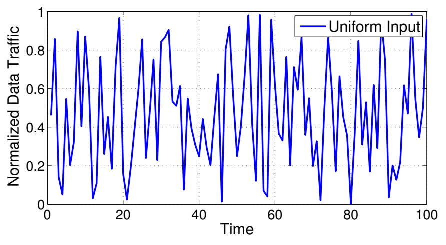

Fig. 2. Uniformly Random Input.

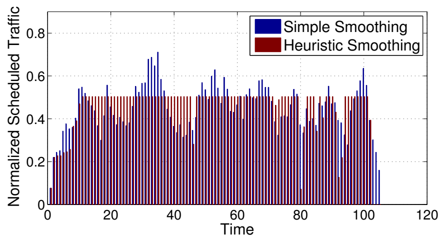

Fig. 3. Simple Smoothing *vs.* Heuristic Smoothing, $D = 10$

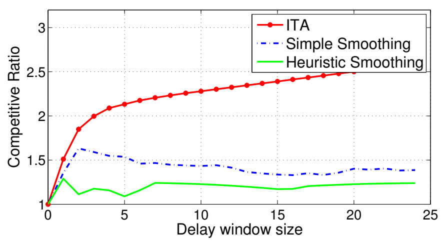

Fig. 4. Competitive ratio over various delay window sizes under input of uniform distribution.

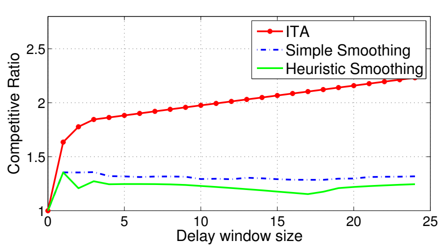

Fig. 5. Competitive ratio over various delay window sizes under input of Poisson distribution.

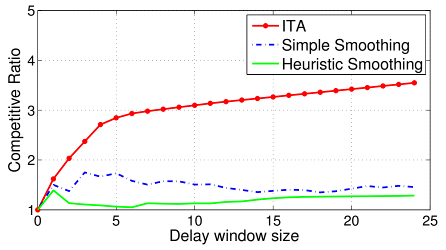

Fig. 6. Competitive ratio over various delay window sizes under input of Gaussian distribution.

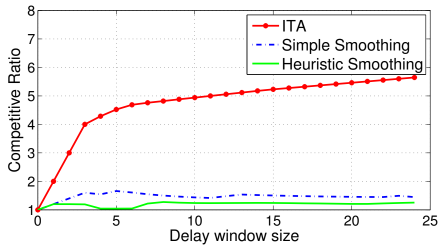

Fig. 7. Competitive ratio over various delay window sizes under a specifically designed input.

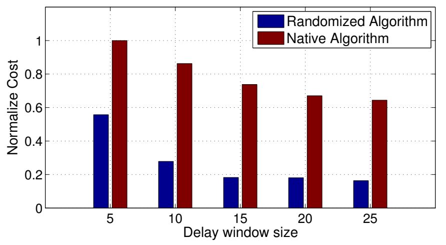

Fig. 8. Comparison between the proposed randomized algorithm and the native algorithm under input of uniform distribution and $\beta = 2$.

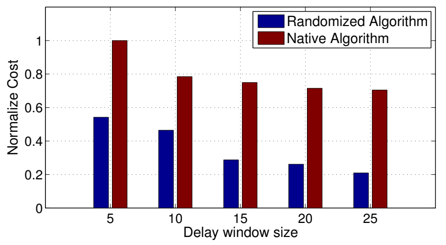

Fig. 9. Comparison between the proposed randomized algorithm and the native algorithm under input of Poisson distribution and $\beta = 2$.

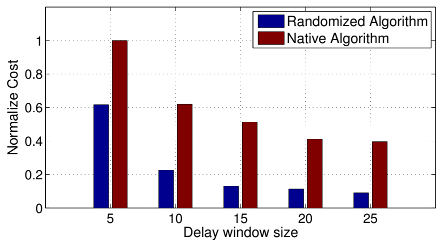

Fig. 10. Comparison between the proposed randomized algorithm and the native algorithm under input of Gaussian distribution and $\beta = 2$.

[4] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational Solutions to Large-scale Data Management and Analysis," *Nat Rev Genet*, vol. 11, no. 9, pp. 647–657, Sep. 2010.

[5] L. Golubchik, S. Khuller, K. Mukherjee, and Y. Yao, "To Send or not to Send: Reducing the Cost of Data Transmission," in *Proc. of IEEE INFOCOM*, 2013.

[6] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. Lau, "Moving Big Data to The Cloud: An Online Cost-Minimizing Approach," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2710–2721, 2013.

[7] H. Wang, H. Xie, L. Qiu, A. Silberschatz, and Y. Yang, "Optimal ISP Subscription for Internet Multihoming: Algorithm Design and Implication Analysis," in *Proc. of IEEE INFOCOM*, 2005.

[8] S. Peak, "Beyond Bandwidth: The Business Case For Data Acceleration," *White Paper*, 2013.

[9] D. K. Goldenberg, L. Qiuy, H. Xie, Y. R. Yang, and Y. Zhang, "Optimizing Cost and Performance for Multihoming," in *Proc. of ACM SIGCOMM*, 2004.

[10] A. Grothey and X. Yang, "Top-percentile Traffic Routing Problem by Dynamic Programming," *Optimization and Engineering*, vol. 12, pp. 631–655, 2011.

[11] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," in *Proc. of IEEE FOCS*, 1995.

[12] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed Scaling to Manage Energy and Temperature," *J. ACM*, vol. 54, no. 1, pp. 3:1–3:39, Mar. 2007.

[13] S. Albers, F. Müller, and S. Schmelzer, "Speed Scaling on Parallel Processors," in *Proc. of ACM SPAA*, 2007.

[14] B. Bingham and M. Greenstreet, "Energy Optimal Scheduling on Multiprocessors with Migration," in *Proc. of IEEE ISPA*, 2008.

[15] E. Angel, E. Bampis, F. Kacem, and D. Letsios, "Speed Scaling on Parallel Processors with Migration," in *Euro-Par 2012 Parallel Processing*, ser. Lecture Notes in Computer Science, C. Kaklamanis, T. Papatheodorou, and P. Spirakis, Eds. Springer Berlin Heidelberg, 2012, vol. 7484, pp. 128–140.

[16] G. Greiner, T. Nonner, and A. Souza, "The Bell is Ringing in Speed-scaled Multiprocessor Scheduling," in *Proc. of ACM SPAA*, 2009.

[17] M. A. Adnan, Y. Ma, R. Sugihara, and R. Gupta, "Dynamic Deferral of Workload for Capacity Provisioning in Data Centers," http://arxiv.org/abs/1109.3839.

[18] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data Centers Power Reduction: A Two Time Scale Approach for Delay Tolerant Workloads," in *Proc. of IEEE INFOCOM*, 2012.

[19] B. Cho and I. Gupta, "New Algorithms for Planning Bulk Transfer via Internet and Shipping Networks," in *Proc. of IEEE ICDCS*, 2010.

[20] M. Adler, R. K. Sitaraman, and H. Venkataramani, "Algorithms for Optimizing the Bandwidth Cost of Content Delivery," *Comput. Netw.*, vol. 55, no. 18, pp. 4007–4020, Dec. 2011.

[21] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[22] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves, "Sailfish: A Framework for Large Scale Data Processing," Yahoo!Labs, Tech. Rep., 2012.

[23] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing MapReduce for Highly Distributed Environments," Department of Computer Science and Engineering, University of Minnesota, Tech. Rep., 2012.

[24] H. Becker and J. Riordan, "The Arithmetic of Bell and Stirling numbers," *American journal of Mathematics*, vol. 70, no. 2, pp. 385–394, 1948.