runData: Re-Distributing Data via Piggybacking for Geo-Distributed Data Analytics Over Edges

Yibo Jin, *Student Member, IEEE*, Zhuzhong Qian, *Member, IEEE*, Song Guo, *Fellow, IEEE*, Sheng Zhang, *Member, IEEE*, Lei Jiao, *Member, IEEE*, and Sanglu Lu, *Member, IEEE*

Abstract—Efficiently analyzing geo-distributed datasets is emerging as a major demand in a cloud-edge system. Since the datasets are often generated in closer proximity to end users, traditional works mainly focus on offloading proper tasks from those hotspot edges to the datacenter to decrease the overall completion time of submitted jobs in a one-shot manner. However, optimizing the completion time of *current job* alone is insufficient in a long-term scope since some datasets would be used multiple times. Instead, optimizing the data distribution is much more efficient and could directly benefit forthcoming jobs, although it may postpone the execution of current one. Unfortunately, due to the throwaway feature of data fetcher, existing data analytics systems fail to re-distribute corresponding data out of hotspot edges after the execution of data analytics. In order to minimize the overall completion time for a *sequence* of jobs as well as to guarantee the performance of current one, we propose to re-distribute the data along with task offloading, and formulate corresponding ε-bounded data-driven task scheduling problem over wide area network under the consideration of edge heterogeneity. We design an online schema runData, which offloads proper tasks and related data via piggybacking to the datacenter based on delicately calculated probabilities. Through rigorous theoretical analysis, runData is proved concentrated on its optimum with high probability. We implement runData based on Spark and HDFS. Both testbed results and trace-driven simulations show that runData re-distributes proper data via piggybacking and achieves up to 37 percent reduction on average response time compared with state-of-the-art schemas.

Index Terms—Cloud-edge system, data re-distribution, heterogeneity, online schema

1 Introduction

Tens of datacenters [2] as well as thousands of nearby edges [3] have already been deployed all over the world by many global companies and organizations, like Google [4], Microsoft [5] and Alibaba [6], in order to provide high-quality services for end users [7]. These nearby edges continuously produce large volume of data, including trillions of user clicks [8], TB-sized logs during the daily usage of diverse applications [9] as well as massive videos recorded for surveillance purposes [10], which is widely used for various timely data analytics [11] and commercial decisions [12].

Aggregating [13] such volume of data from nearby edges to the datacenter over wide area network (WAN) [12] easily incurs long transmission delays. As a result, previous works mainly focus on those network-aware strategies to shorten the transmission delay over WAN [9], [12], [14]. That is, in order to decrease the volume of data transmissions, those works prefer to queuing adequate data analytics tasks at nearby edges [15], [16], [17] while offloading the rest to the datacenter. Since a submitted job often contains multiple tasks [15], [18] and only the completion of the straggliest one decides its termination [9], these works essentially minimize the maximal completion time of all tasks for each submitted job, but the optimization is conducted in a one-shot manner.

However, optimizing the completion time for each submitted job in a one-shot manner or an on-demand manner is insufficient. First, since some datasets generated would be used multiple times [19], [20], offloading those hot data from edges to the datacenter directly benefits forthcoming jobs. Unfortunately, those one-shot or on-demand strategies are unwilling to re-distribute the hot data because it would postpone the execution of current jobs. Second, as shown in our system analysis later, due to the throwaway feature of the data fetchers in existing data analytics systems [21], transferred data stored in memory is discarded during the execution of the task, and the data distribution is essentially unchanged. Third, the data is often generated unevenly across geo-distributed edges [18], skewed data distribution easily overloads those edges with poor computing capacities [16], [17] repeatedly, or even leads to the outages [22]. Therefore, for the overall performance of the data analytics system, we prefer to re-distribute the data from heterogenous geo-distributed edges to the datacenter, in order to minimize the overall completion time for a sequence of jobs.

Unfortunately, re-distributing hot data out of hotspot edges as early as possible is non-trivial. On the one hand, traditional approaches, like detecting the hot data upon the

- *Yibo Jin, Zhuzhong Qian, Sheng Zhang, and Sanglu Lu are with the State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing Jiangsu 210023, China. E-mail: yibo.jin@smail.nju.edu.cn, {qzz, sheng, sanglu}@nju.edu.cn.*
- *Song Guo is with the Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong, and also with the Hong Kong Polytechnic University Shenzhen Research Institute, Shenzhen, Guangdong 515063, China. E-mail: song.guo@polyu.edu.hk.*
- *Lei Jiao is with the Department of Computer and Information Science, University of Oregon, Eugene, OR 97403 USA. E-mail: jiao@cs.uoregon.edu.*

Manuscript received 10 Nov. 2020; revised 27 Apr. 2021; accepted 28 May 2021. Date of publication 3 June 2021; date of current version 28 June 2021.
(Corresponding author: Zhuzhong Qian.)
Recommended for acceptance by Q. Zheng.
Digital Object Identifier no. 10.1109/TPDS.2021.3086274

number of their accesses heuristically, are unable to classify those target candidates until the access counter reaches to a pre-defined threshold. Then, hot data has to be crowded within hotspot edges before the actual data re-distribution. On the other hand, the bandwidth over WAN is often costly and limited [18]. Aggregating large volume of data from edges to the datacenter is unrealistic for timely commercial data analytics. Furthermore, there is a conflict between re-distributing hot data as early as possible and postponing a little on the execution of the job currently submitted.

Inspired by the fact that each task has to fetch its required data before execution, in order to avoid discarding already transferred data in existing data analytics systems, we propose to keep it and validate it in the datacenter along with task offloading and execution, i.e., data re-distribution via piggybacking in this paper. As a result, on the one hand, thanks to hot data re-distribution, instead of being crowded in hotspot edges, forthcoming tasks can be directly executed within the datacenter, leading to a lower completion time. On the other hand, edges have less opportunities to be overloaded since the data required by those computing-intensive tasks is more likely to be offloaded to the datacenter.

We address the issue of re-distributing the data between heterogeneous geo-distributed edges and the datacenter by formulating ε-bounded data-driven task scheduling problem, to minimize the overall completion time for a sequence of jobs, as well as to ensure the performance of currently submitted job and WAN usage controlled by ε. Afterwards, we propose an online schema runData, offloading proper tasks while re-distributing related data from the edges to the datacenter via piggybacking based on delicately calculated probabilities. Specifically, we use the probabilities calculated as the preference to assign tasks and decide the distribution.

Through rigorous theoretical analysis, runData can be proved concentrated on its optimum with high probabilities by using Martingale analysis. Both testbed results and trace-driven simulations show that runData reaches up to 37 percent reduction regarding the average job completion time by transferring hot data candidates as early as possible. More specifically, the overall accesses of nearly 40 percent data uploaded by runData are higher than 5 and nearby 90 percent of uploaded data is transferred when their accesses are just less than 4.

To the best of our knowledge, it is the first to propose and design data-driven task scheduling via piggybacking in a cloud-edge system. Our contributions are summarized as

- We formulate ε-bounded data-driven task scheduling problem (εD-GeoTS), which essentially optimizes the overall latency for a sequence of jobs as well as guarantees the response time for each job and WAN usage by re-distributing data via piggybacking.
- We design an online schema runData, which offloads proper tasks as well as re-distributes corresponding data to the datacenter via piggybacking. By rigorous theoretical analysis, runData can be proved concentrated on its optimum with high probability.
- We implement a prototype based on Spark and HDFS and evaluate runData with trace-driven simulations. The results show that runData achieves up to 37 percent reduction compared with state-of-the-art schemas.

2 Related Work

We summarize prior research in three categories, and highlight their drawbacks compared to our work, respectively.

2.1 Intra-Cluster Data Management

Previous works regarding the management of data within a cluster focused on two aspects: 1) profiling the characteristics of data analytics according to their usage patterns on the resources; 2) conducting the management of resources to balance the skewed workloads over datasets.

Note that precisely predicting the data access patterns for various data analytics was shown difficult due to their diverse functionalities [23], [24]. Mantri [25] and Grass [26] studied the relationship between task durations and some dynamic factors, including the congestion of I/O, the inference, etc., among concurrent tasks. Graphene [23] profiled DAGs for better packing strategy. Some works also investigated on the management of data for imbalanced usage on datasets. Datanet [27] focused on managing the datasets by using an elastic structure. NearestFit [28] designed a novel profile-guided progress indicator, which predicted the data skewness and the stragglers to avoid the excessive use of resources. SkewTune [29] balanced the usage of computing resources for various workloads and related datasets. Chen *et al.* [30] studied a hybrid approach combing data-parallel and task-parallel optimization. Other works, like [31], [32], [33], discussed task-parallel executions for data analytics.

Unfortunately, when large volume of data is generated at the end of the network, previous works fail to treat the costly transmissions over WAN and skewed data distribution via the data re-distribution along with the task offloading.

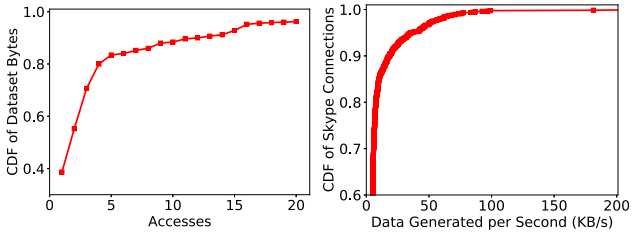
2.2 Inter-Cluster Management on Resources

Existing works mainly focused on reducing the transmission of data over WAN for data analytics. JetStream [13] aimed at aggregating the data to one site, e.g., a cluster or the datacenter, under the consideration of insufficient bandwidths. Geode [9] studied the optimization of WAN usage among geo-distributed sites. ADP [35] used hypergraph to model the user requests, and split it with the least cost. Iridium [12] optimized the proportions of tasks to different sites in order to avoid the bottleneck links, and heuristically moving the data out of bottleneck edges in advance. Flutter [15] and SWAG [16] studied both network transmission and computation for concurrent tasks among sites, in order to optimize the straggliest. To deal with both heterogeneous computing capacity and WAN bandwidth, Tetrium [18] was proposed for speeding up overall latency of data analytics.

Although precious works have already considered both computation and transmission, their optimization objectives only focus on those current jobs. Instead, we design a data-driven task scheduling for jobs in an online manner.

2.3 Resource Management at Edges

Other works optimized the resource usage for data analytics and corresponding applications at edges. Some of them tried to propose resource allocation schemas for video analytics. DIVS [36] proposed task-level parallel and model-level parallel training models to accelerate the analytics of videos. VideoStorm [10], [37] accelerated video analytics queries on live



(a) Popularity of accessed data (bytes) in Microsoft

(b) Variation on generated data sizes from Skype

Fig. 1. Preliminary case studies: (a) Popularity of accessed bytes from Cosmos [20]; (b) Variation on generated data sizes from Skype [34].

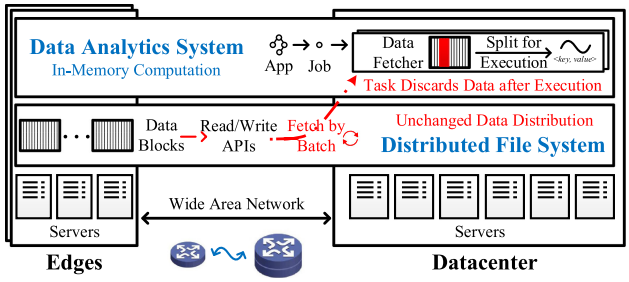


Fig. 2. Throwaway data fetcher in existing system batch by batch.

video streams. And the rest studied the deployment of machine learning models. DADS [38] optimally partitioned the DNN under different network conditions. IONN [39] designed a partitioning-based DNN offloading schema.

Those works conduct the resource management at edges for various applications, but they fail to offload the tasks with data re-distribution to avoid heavy workloads at edges.

3 Motivation and System Model

3.1 Motivation for Re-Distribution via Piggybacking

Massive data is often generated within nearby edges, easily leading to heavy workloads on those hotspots. Since some data would be used multiple times, offloading computing-intensive tasks and re-distributing related hot data out of these bottleneck edges to the datacenter as early as possible would benefit forthcoming jobs. Unfortunately, existing data fetchers embedded in those data analytics systems fail to re-distribute the input data along with the task offloading since the data is fetched batch by batch in a throwaway manner.

Frequently Used Geo-distributed Datasets. As shown in previous studies, many datasets are accessed by multiple analytics over time. For example, as high as 721 accesses are achieved within 24 hours in Facebook Trace [19]. Further shown in Fig. 1a, 11.6 percent datasets are accessed for more than 10 times in Microsoft Cosmos [20]. Moreover, such datasets are often unevenly generated across geo-distributed edges, near to those end users. For example, based on the analysis of Skpye logs over 126 Azure sites, relative to the site with the minimum data generated, the median, 90th percentile and the maximum values are 8x, 15x and 22x, respectively, more [18]. Similar results are also illustrated in [34], where the data volumes generated per second derived from 1249 Skype connections are quite different, as shown in Fig. 1b.

Easily Overloaded Heterogeneous Edges. With the cumulative process of data accesses, overloaded datasets are more likely to be hot data candidates and would be accessed by data analytics again. Restricted to limited resources [40], [41] at edges, frequent accesses on the datasets easily incurs heavy workloads and overloads geo-distributed edges. Furthermore, the computing resources consumed by different data analytics are quite different [15], [23]. Then, the datasets related to computing-intensive tasks are more likely to result in hotspots at edges, especially when the heterogeneity [17], [18] occurs. As a consequence, the completion time of related data analytics, executed within such overloaded edges and hotspots, would be no doubt elongated.

Throwaway Feature of Data Fetcher in Analytics. Each data analytics task treats the raw input data as follows [21], as shown in Fig. 2: 1) the data fetcher connects the distributed file system, e.g., Hadoop Distributed File System (HDFS) [42], and requires the raw input data batch by batch, e.g., 64KB sized data for a batch; 2) the LineReader embedded in the task splits each batch data into multiple lines according to predefined line delimiters, e.g., CR ('\r') and LF ('\n'); 3) the TextInputFormat embedded in the task further splits each line string into multiple $<key, value>$ pairs according to the data record delimiters, e.g., the comma; and 4) finally, the task treats each data record for data analytics.

Note that all of these operations are conducted in memory. As long as the batch data has already been fetched and split for data analytics task, it would be discarded directly, i.e., the throwaway feature of the data fetcher. Essentially, distributed file systems are designed to support the pipeline execution for data analytics tasks. As a result, the raw input data, i.e., the data block, is fetched batch by batch through Read APIs after the establishment of connections. After the executions of all tasks, the data distribution is essentially unchanged, i.e., the data is still stored within the edges near to the end users, unless the distributed file system balances the data itself according to the data volume.

Need for Re-Distribution via Piggybacking. On the one hand, re-distributing hot data candidates as early as possible can intuitively release the heavy workload on hotspot edges, decreasing the completion time of incoming jobs. Actually, a task is offloaded to the datacenter means related input data needs to be transferred either. Thus, keeping those hot data candidates within the datacenter has no extra bandwidth cost over WAN, as shown in Fig. 3. Data re-distribution via piggybacking just keeps in store already transferred data in memory batch by batch after the completion of each task, which does little harm to the execution of data analytics. On the other hand, those data related to computing intensive tasks should also be transferred to the datacenter as early as possible along with the migration of tasks.

Local Optimum versus Global Optimum. The global objective is no doubt to improve the performance of entire data analytics system, i.e., minimizing the overall completion time for jobs in a long-term scope. Although the data re-distribution actually speeds up the forthcoming jobs, it may also defer a little bit on currently submitted jobs for better global performance based on the task offloading strategy. A typical example is that a dataset will be multiply used by several jobs, and the global optimum solution is to transfer

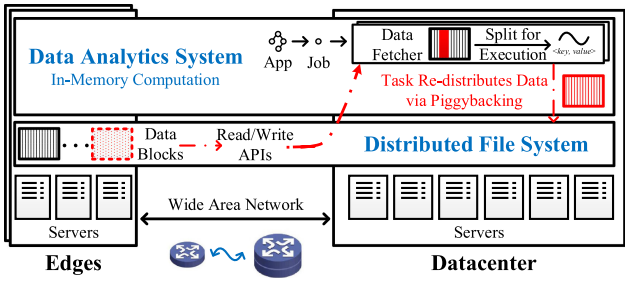


Fig. 3. Data re-distribution via piggybacking along with task offloading.

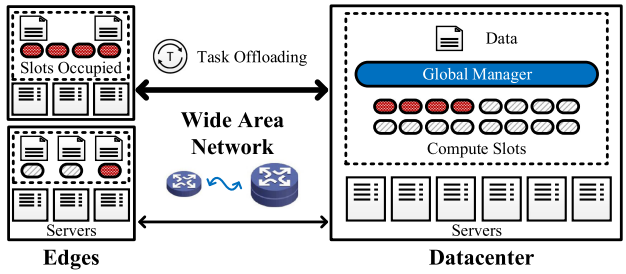


Fig. 4. Modeling for the data analytics in a cloud-edge system.

the whole dataset to the datacenter at the very beginning. As a result, the first job has to be delayed due to long data transmission over WAN, which should be also avoided.

3.2 System Model

Table 1 illustrates the main notations used. A typical cloud-edge system, as shown in Fig. 4, consists of the datacenter and multiple heterogeneous edges, offering various services to users based on some commonly used big data analytics frameworks, like Hadoop [43] and Spark [21]. Such frameworks not only manage the resources, but also schedule the tasks and track the status for submitted jobs, i.e., the local manager embedded within each edge keeps tracking of available local resources and task status [21], [43], as well as periodically updates the global manager in the datacenter.

Job Description. For each job submitted to data analytics system, the global manager often converts it into a Directed Acyclic Graph (DAG) of stages [12], defined as $G = (\mathcal{V}, \mathcal{E})$, where each edge $e \in \mathcal{E}$ represents the dependency, and each node $v \in \mathcal{V}$ represents a stage containing many concurrent tasks with similar functionality [9], [15]. Although scheduling the DAGs over WAN has already been widely studied by previous works [9], [23], most of them largely depend on the estimation for subsequent untreated stages. Thus, for computation feasibility, we only consider the scheduling of concurrent tasks for each stage step by step according to their dependencies. We denote by $\mathcal{T}_j$ the task set pending for scheduling in current stage of job-j. Note that each stage of original submitted jobs is a new job instead in this paper to the data analytics system like existing systems [21], [43] and previous works [15], [18], and $\forall j_1, j_2, \mathcal{T}_{j_1} \cap \mathcal{T}_{j_2} = \varnothing$.

TABLE 1
Summary of Main Notations Used

Symbol	Description
$\mathcal{G}$	Set of all edges considered in the cloud-edge system
$\mathcal{T}_j$	Task set of job-j, i.e., consists of multiple concurrent tasks
e_k	Execution time of task-k in the datacenter, $k \in \mathcal{T}_j$
γ_i	The ratio of CPU frequency at edge-i to that of DC[1]
$\mathcal{S}_i$	Set of compute slots hosted by edge-i
h_{si}	Binary variable indicates whether edge-i hosts slot-s
$\mathcal{L}_k$	Set of edges with required data stored for task-k, $\mathcal{L}_k \subseteq \mathcal{G}$
	Primary copy for each data block, i.e., $\lvert\mathcal{L}_k\rvert = 1$
$\mathcal{R}_{js}$	Tasks queued on slot-s in front when job-j is submitted
q_{js}	Queue delay for task-k if it is assigned to slot-s, $k \in \mathcal{T}_j$
$\mathcal{N}_{ji}$	Set of tasks in job-j offloaded from edge-i to DC
σ_i	Transmission delay for one data block from edge-i to DC
b_k	Transmission delay of task-k from edge-i to DC, $i \in \mathcal{L}_k$
C_j	Completion time of job-j, depends on the straggliest task
Decision	**Description**
I_{ks}	Indicator of assigning task-k to slot-s

1. *The term 'DC' is the abbreviation of datacenter.*

Data and Edge Description. Each task requires one data block with uniform size, e.g., 128MB, stored in distributed file system, e.g., HDFS [42] for execution. Note that we only consider the primary copy for each block [16] under the consideration of the overhead for maintaining the consistency among multiple copies. To describe the computing capacity of geo-distributed edges, e.g., cellular base stations with co-located servers [44], we use the compute slot as the minimal resource unit [45], including memory, CPU, etc. [21], [46], each of which has a local queue for tasks [47] if the slot is currently in use. We denote by $\mathcal{S}_i$ the set of slots hosted by edge-i, $i \in \mathcal{G}$, and denote by $\mathcal{G}$ the set of all edges. Then, the variable I_{ks} indicates whether task-k, $k \in \mathcal{T}_j$ for job-j is assigned to slot-s, $s \in \mathcal{S}_i$ hosted by edge-i, or not.

Task Execution. Generally, the datacenter has adequate compute slots and better computing ability, so that the tasks deployed to the datacenter run faster than that at edges [48]. In this paper, we define the base execution time for task-k, i.e., e_k as its running time in the datacenter. The estimation accuracy can reach up to 80 percent [26], through estimating the task duration from those finished ones. Afterwards, we use γ_i (≥ 1) to represent the performance difference between the datacenter and edge-i. Note that the performance difference is mainly caused by the difference on CPU [49], including the number of threads per core and its basic frequency. Then, the execution time of task-k at edge-i is $\gamma_i e_k$ accordingly. If task-k is scheduled to slot-s at edge-i, its queuing time, represented as q_{js}, is the sum of all expected execution time of all tasks queued in front, based on the FIFO strategy [14], when job-j is submitted, $k \in \mathcal{T}_j$. That is

$$\forall j, s : q_{js} \triangleq \sum_{k' \in \mathcal{R}_{js}, s \in \mathcal{S}_i, i \in \mathcal{L}_{k'}} h_{si} \gamma_i e_{k'},$$

where $\mathcal{R}_{js}$ is the set of all tasks queued in front of task-k on slot-s after the submission of job-j, $k \in \mathcal{T}_j$. We use h_{si} to indicate whether slot-s is hosted by edge-i, $i \in \mathcal{L}_{k'}$, and $\mathcal{L}_{k'}$ indicates the set of edges with required input data block for

task-k'. Note that h_{si} is the binary variable already known before scheduling, and $|\mathcal{L}_k| = 1, \mathcal{L}_k \subseteq \{\mathcal{G} \bigcup \{DC\}\}$ only considers the primary copy for each data block stored and used as mentioned before, where $\{DC\}$ indicates that the related data block is stored within the datacenter.

Task Transmission. Each edge connects to the datacenter with limited WAN bandwidth, which follows the strategy of shared usage [50], i.e., those concurrently data transmissions share the bandwidth equally. For the tasks offloaded to the datacenter over WAN, they need to fetch their required data blocks before execution, although the fetcher actually requires related data blocks in a batch manner as mentioned before. Here, we denote by σ_i the data transmission delay for transferring only one data block with uniform size from edge-i to the datacenter with fully use of current available bandwidth. Due to the shared usage on the bandwidth, the actual data transmission delay naturally relies on the number of concurrent tasks offloaded to the datacenter. Then, the data transmission delay b_k for any offloaded task-k, whose required data block is stored in edge-i, is

$$\forall j, k : b_k \triangleq |\mathcal{N}_{ji}|\sigma_i, \quad i \in \mathcal{L}_k, k \in \mathcal{T}_j,$$

where $\mathcal{N}_{ji}$ is the set of tasks offloaded out of edge-i.

3.3 Problem Formulation

The task scheduler embedded in the global manager assigns all of the tasks, corresponding to submitted jobs, either at edges or to the datacenter, to minimize the overall completion time. The overall completion time of job-j, defined as C_j, relies on the straggliest task over those ones assigned at edges and those ones offloaded to the datacenter, i.e.,

$$\forall j : C_j \triangleq \max\{Local_j, Remote_j\},$$

where $Local_j$ and $Remote_j$ represent the completion time of the straggliest task assigned at edges and offloaded to the datacenter, respectively. On the one hand, the completion time of the straggliest task assigned at edges depends on the queue condition and its execution, i.e.,

$$\forall j : Local_j \triangleq \max_{s \in \{\cup \mathcal{S}_i, i \in \mathcal{G}\}} \{q_{js} + \sum_{k \in \mathcal{T}_j, i \in \mathcal{L}_k} I_{ks} h_{si} \gamma_i e_k\},$$

where $\{\cup \mathcal{S}_i, i \in \mathcal{G}\}$ considers all of the slots over edges.

On the other hand, the completion time of those tasks offloaded from edges to the datacenter also depends on the straggliest in terms of both the transmission over WAN and the execution in the datacenter, i.e.,

$$\forall j : Remote_j \triangleq \max_{k \in \mathcal{T}_j} \{X_k \cdot (b_k + e_k)\},$$

where the variable $X_k \in \{1, 0\}$ indicates whether task-k is offloaded or not. Note that X_k can be represented by the decisions. The relationship between X_k and $\{I_{ks}\}$ is

$$\forall k : X_k \triangleq 1 - \sum_{s \in \mathcal{S}_i, i \in \mathcal{L}_k} I_{ks}.$$

Actually, the previous equation guarantees that a task can only be assigned either to a slot hosted by the edge, in which required data block is stored, or directly offloaded to the datacenter. For those tasks offloaded to the datacenter, i.e., $X_k = 1$, there is no need to queue them any more since the datacenter has better computing capacity than that of edges as mentioned in previous subsection.

One-Shot Task Offloading. GeoTS, the Geo-distributed Task Scheduling Problem for current job-j, similar to those works [15], [18], essentially offloading tasks between edges and the datacenter, is to minimize the overall task completion time, i.e., $C_j \triangleq \max\{Local_j, Remote_j\}$ as follows:

$$\begin{aligned} min \quad & C_j & [\mathrm{G}eoTS] \\ s.t. \quad & X_k \triangleq 1 - \sum_{s \in \mathcal{S}_i, i \in \mathcal{L}_k} I_{ks}, \quad X_k \in \{1, 0\}, \quad \forall k, \\ var. \quad & I_{ks} \in \{1, 0\}, \quad \forall k, s. \end{aligned}$$

Other works also focus on the management of tasks with diverse objective forms. For example, some models [51], [52] in terms of the task transmissions over wireless networks and edge networks are studied, cumulative task completion time is considered [53], [54], etc. However, the inner challenges for the problem in terms of the assignment of tasks are unchanged. That is, the decisions for the assignment of tasks are often discrete and the related problems are then NP-hard. Note that the property of NP-hard is used to describe the decision versions of these proposed optimization problem, in order to show how hard the proposed problems are, which hampers us from efficient solutions.

We use $Opt(C_j)$ here to represent the optimum of GeoTS. However, optimizing current job-j is sub-optimal because each single job submitted is unwilling to spend extra time or cost for data re-distribution according to the objective form in the GeoTS. As a result, hot data has to be crowded within hotspot edges, elongating the completion time of incoming related jobs because some datasets would be used multiple times. Therefore, in terms of the overall completion time for a sequence of jobs, the global optimum should be achieved by both task offloading and data re-distribution.

Task Offloading for Jobs via Piggybacking. The objective of the global optimum is to minimize the overall latency for a sequence of jobs, via task offloading along with the data re-distribution between the edges and the datacenter (DC), i.e., $\{I_{ks}\}$ also indicates the data locations. We have

$$\begin{aligned} min \quad & \sum_j C_j & [Global - \mathrm{G}eoTS] \\ s.t. \quad & \mathcal{L}_k = \{\mathrm{DC}\} \ \mathrm{if} \ \ X_k = 1, \quad \forall k, \\ & X_k \triangleq 1 - \sum_{s \in \mathcal{S}_i, i \in \mathcal{L}_k} I_{ks}, \quad \forall k, \\ var. \quad & I_{ks} \in \{1, 0\}, \quad \forall k, s. \end{aligned}$$

However, achieving the global optimum may also defer the completion time of each single job, which is unrealistic for timely data analytics. Hence, we try to balance the global optimum as well as the tolerable completion time for current job, i.e., task offloading via piggybacking should be closer to its local optimum. Therefore, we formulate ε-bounded data-driven task scheduling problem as follows:

Definition 1. (εD-GeoTS) ε-bounded Geo-distributed Data-Driven Task Scheduling Problem. *With the system models, we formulate the following optimization problem:*

$$\begin{aligned} min \quad & \sum_j C_j && [\varepsilon D - GeoTS] \\ s.t. \quad & Remote_j \le \mathcal{F}(Opt(C_j), \varepsilon_j), \ \forall j, && (0) \\ & \boldsymbol{\mathcal{L}_k} = \{\mathrm{DC}\} \ \text{if} \ X_k \triangleq 1 - \sum_{s \in \boldsymbol{\mathcal{S}_i}, i \in \boldsymbol{\mathcal{L}_k}} I_{ks} = 1, \ \forall k, \\ var. \quad & I_{ks} \in \{1, 0\}, \ \forall k, s, \end{aligned}$$

where function $\mathcal{F}$ takes the local optimum and ε_j as its input.

Unfortunately, evenly the problem of solving the local optimum $Opt(C_j)$ in εD-GeoTS, i.e., solving GeoTS first, has already been proved as NP-hard. Due to the infeasibility of $Opt(C_j)$, we propose to use its feasible lower bound as the substitute shown later, which also facilitates our analysis and system implementation, upon the monotonicity of $\mathcal{F}$.

$\mathcal{F}$ has multiple feasible forms. However, the most intuitive form is linear, i.e., $\mathcal{F}(Opt(C_j), \varepsilon_j) \triangleq Opt(C_j) + \varepsilon_j$, since ε_j directly shows the distance away from the local optimum of job-j. Actually, any monotone function $\mathcal{F}$ respect to both $Opt(C_j)$ and ε_j is meaningful. Note that, given $Opt(C_j)$ or its substitute, if $\mathcal{F}$ is linear or quadratic respect to the second input ε_j, the computational complexity of solving a problem with a linear objective and Constraint (0) is fairly low [55] in the domain of reals, and closed-form solutions are available. Nevertheless, if $\mathcal{F}$ is generally a convex function, solving the problem with Constraint (0) alone comes with a higher computational complexity to obtain the solutions.

Then, the linear form of $\mathcal{F}$ adopted in this paper is the most suitable one for both tractability and interpretability.

Algorithm 1. Online Schema *run*Data

// Triggered for each submitted job in an online manner.
// **Step 1** solves proposed *lp*-GeoTS for submitted job-j.
1: Local*Opt*_Based_Task_Offloading (Algorithm 2);

// **Step 2** makes the decisions for data re-distribution.
2: ε_Bounded_Data_Driven_Task_Scheduling (Algorithm 3).

4 Online Schema Design

The proposed online data-driven schema *run*Data in this paper solves εD-GeoTS within polynomial time based on two key steps, as shown in Algorithm 1:

i) *Algorithm 2* generates Local*Opt*-based task offloading using the results solved from the relaxed version of GeoTS, i.e., *lp*GeoTS, as well as using randomized task assignment, which facilitates the theoretical analysis later;
ii) *Algorithm 3* further conducts the ε-bounded data re-distribution upon Local*Opt*-based task offloading, which re-distributes extra data blocks via idle bandwidth before the completion of the straggliest with controlled performance.

The relationship between all of the problems proposed and the algorithms are illustrated in Fig. 5. Actually, balancing the workloads, controlling the network communication overhead and fully utilizing the task synchronization in the system are key issues involved in the assignment of tasks. *run*Data is a well-designed online schema for these issues.

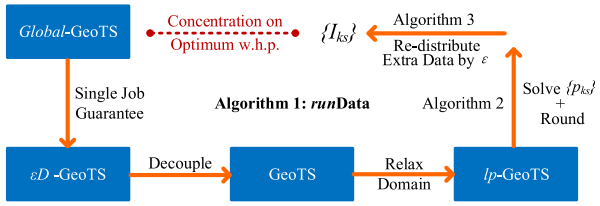


Fig. 5. Relationship between proposed problems and algorithms.

4.1 Local*Opt*-Based Task Offloading

To obtain the feasible substitute for $Opt(C_j)$ in εD-GeoTS, we rely on its relax version, i.e., *lp*-GeoTS shown later, and use it to guide the scheduling of εD-GeoTS. The optimum of *lp*-GeoTS actually shows the theoretical lower bound of GeoTS due to relaxed domains of related decisions.

Problem Transformation. Since solving GeoTS for the optimum $Opt(C_j)$ is infeasible if NP $\neq$ P, we try to relax the domain of GeoTS from integers to reals as follows:

$$\begin{aligned} min \quad & \widetilde{C}_j && [lp - GeoTS] \\ s.t. \quad & \widetilde{X}_k \triangleq 1 - \sum_{s \in \boldsymbol{\mathcal{S}_i}, i \in \boldsymbol{\mathcal{L}_k}} p_{ks}, \ \widetilde{X}_k \in [0, 1], \ \forall k, \\ var. \quad & p_{ks} \in [0, 1], \ \forall k, s, \end{aligned}$$

where we substitute variables $I_{ks} \in \{1, 0\}$ and $X_k \in \{1, 0\}$ for reals $p_{ks}, \widetilde{X}_k \in [0, 1], \forall k$ in order to obtain $\widetilde{C}_j$, as well as conduct related changes in the definition of C_j for GeoTS.

Correspondingly, we denote by $Opt(\widetilde{C}_j)$ the optimum solved from *lp*-GeoTS. Since the optimum of *lp*-GeoTS (under real domain) is better than any other feasible solution for GeoTS (under integer domain), we have

$$\forall j: \ \ Opt(\widetilde{C}_j) \le Opt(C_j).$$

Algorithm 2. Local*Opt*_Based_Task_Offloading

Require: Set of tasks requested by job-j: $\boldsymbol{\mathcal{T}_j}$;
Transmission delay between edges and DC: $\{\sigma_i\}$;
Performance difference among edges: $\{\gamma_i\}$;
Base execution time for tasks: $\{e_k\}$.
1: $Opt(\widetilde{C}_j), \{p_{ks}\} \leftarrow$ Solve *lp*-GeoTS;
2: **for** each task-$k \in \boldsymbol{\mathcal{T}_j}$ **do**
3: $\{I_{ks}\}_1$ = Randomized_Rounding($\{p_{ks}\}$),
$\{I_{ks}\}_2$ = Randomized_Rounding($\{p_{ks}\}$);
4: **end for**
5: $\{C_j\}_1$ = GeoTS($\boldsymbol{\mathcal{T}_j}, \{\gamma_i\}, \{\sigma_i\}, \{e_k\}, \{I_{ks}\}_1$),
$\{C_j\}_2$ = GeoTS($\boldsymbol{\mathcal{T}_j}, \{\gamma_i\}, \{\sigma_i\}, \{e_k\}, \{I_{ks}\}_2$);
6: $\Omega_j = \min\{\{C_j\}_1, \{C_j\}_2\}$;
7: Return the values to Algorithm 1: $\Omega_j, \{I_{ks}\}, Opt(\widetilde{C}_j)$.

Thus, we substitute $Opt(C_j)$ in εD-GeoTS for $Opt(\widetilde{C}_j)$ due to the feasibility of $Opt(\widetilde{C}_j)$ by using the mature linear programming techniques. Actually, $Opt(\widetilde{C}_j)$ shows the lower bound of $Opt(C_j)$. Thus, we use it to control the WAN usage and the transmission delay. Then, we have

$$\forall j: \ \mathcal{F}(Opt(\widetilde{C}_j), \varepsilon_j) \triangleq Opt(\widetilde{C}_j) + \varepsilon_j,$$
$$\forall j: Remote_j \leq \mathcal{F}(Opt(\widetilde{C}_j), \varepsilon_j) \leq \mathcal{F}(Opt(C_j), \varepsilon_j).$$

Algorithm 3. ε_Bounded_Data_Driven_Task_Scheduling

Require: $Opt(\widetilde{C}_j)$ and Ω_j from Algorithm 2;
LocalOpt based task offloading: $\{I_{ks}\}$.
1: $\varepsilon_j = \min_{i,k}\{\sigma_i, \gamma_i e_k\}$;
2: $\beta_j = Opt(\widetilde{C}_j) + \varepsilon_j - \Omega_j$;
3: **for** each edge-i **do**
4: Re-distributing data blocks in: $\min\{\max\{0, \beta_j\}, \Theta\}$;
5: Update $\{I_{ks}\}$ for re-distributing extra data blocks;
6: **end for**
7: Deploy tasks with data re-distribution using $\{I_{ks}\}$.

Randomized Task Assignment. $\{p_{ks}\}$ and $\{\widetilde{X}_k\}$ can be seen as a guidance for scheduling since it actually shows the preference on the task assignment based on current system status. Thus, we use $\{p_{ks}\}$ solved as a series of probabilities to assign each task between the edges and the datacenter.

To apply these solved probabilities to a feasible solution of GeoTS, shown in line 3 of Algorithm 3, we use the randomized rounding strategy on $\{p_{ks}\}$ and $\{\widetilde{X}_k\}$ as follows:

i) $\forall k \in \boldsymbol{\mathcal{T}_j}$, we randomly pick a decimal $\xi_k \in (0, 1]$;
ii) $\forall s \in \boldsymbol{\mathcal{S}_i}, i \in \boldsymbol{\mathcal{L}_k}$, I_{ks} equals 1 if and only if ξ_k falls into the interval $(\sum_{s' \in \boldsymbol{\mathcal{U}_s}} p_{ks'}, \sum_{s' \in \boldsymbol{\mathcal{V}_s}} p_{ks'}]$, otherwise I_{ks} equals 0. $\boldsymbol{\mathcal{U}_s}$ contains all of the slots, whose slot indexes are less than that of s; and $\boldsymbol{\mathcal{V}_s}$ contains all of the slots, whose slot indexes are less than or equal to that of slot-s.

Actually, given k, a series of variables $\{I_{ks}\}$ split the interval $(0, 1]$ into multiple parts. And only when ξ_k falls into the interval corresponding to slot-s, I_{ks} equals 1, and the interval length for slot-s is exactly p_{ks}. Thus, we have

$$\forall k, s: E[I_{ks}] = p_{ks}.$$

Then, $\{X_{ks}\}$ can be directly obtained based on its definition in GeoTS. Essentially, the higher value of p_{ks} shows much more preference on slot-s regarding the assignment.

Power of Two Choices. Although each task is deployed according to a series of probabilities solved based on current system status, i.e., based on $\{p_{ks}\}$, the bad event in terms of long completion time for a single job may still occur with a certain probability by using rounded $\{I_{ks}\}$. In order to avoid such bad event as much as possible, we conduct the power of two choices, as shown in line 6 of Algorithm 2. That is, we use the randomized rounding twice upon $\{p_{ks}\}$, to obtain $\{I_{ks}\}_1$ and $\{I_{ks}\}_2$, and choose the better one in terms of the lower completion time for GeoTS. We denote by Ω_j the lower completion time after adopting the power of two choices in our designed randomized task assignment.

Remark 1. First of all, when multiple jobs are submitted, runData is actually triggered for each job in order of their submitted timestamps, just like Hadoop [43] and Spark [21]. And after the assignment for a specific job, the descriptions of related tasks are added to the queues of related slots, e.g., the Executors in Spark. Note that the assignment takes effect directly after the execution of runData, like Spark, instead of periodically updating the status, like Hadoop, since the assignment only transfers those meta information of tasks, i.e., task descriptions. Further transmissions regarding input data blocks and the codes for these tasks are conducted only after the trigger of runnable task processes by the executors.

Remark 2. Although for those computing-intensive tasks, the memory may be insufficient, the memory kept and used for the data re-distribution via piggybacking is essentially fixed, i.e., the size of one data block, since a task uses one block as its input. Such memory kept for data re-distribution could be further compressed if the excessive part is flushed out asynchronously, which does little harm to the execution.

4.2 ε-Bounded Data-Driven Task Scheduling

Although LocalOpt-based task offloading actually transfers those hot data candidates and those computing-intensive tasks to the datacenter through the optimization of GeoTS, the termination of a job is decided by the straggliest task.

Wait for the Straggliest. All of the edges, expect for the one hosting the straggliest, can fully utilize the waiting for re-distributing extra data blocks before the completion of a job. Via piggybacking, current job-j uses extra but controlled time lag, i.e., ε_j for further data re-distribution to benefit forthcoming jobs. As shown in line 1 of Algorithm 3, current status of the cloud-edge system can be seen as the preference on data re-distribution. Therefore, ε_j is carefully chosen as $\min_{i,k}\{\sigma_i, \gamma_i e_k\}$, which also facilitates the analysis later. More specifically, for each job-j submitted in an online manner, we re-distribute $\lfloor \min\{\Theta, \max\{0, \beta_j\}\}/\sigma_i \rfloor$ more data blocks from edge-i to the datacenter, as shown in lines 2 and 4 of Algorithm 3, where $\beta_j = Opt(\widetilde{C}_j) + \varepsilon_j - \Omega_j$ shows the extra but controlled time lag for WAN usage. Note that Θ here represents the minimum inter-arrival time [16] between two consecutive jobs. That is, if the edges are busy, extra data re-distribution is unnecessary and cancelled.

Extra Offloaded Data. Essentially, during the extra data re-distribution before the completion of the straggliest, we prefer to re-distribute those hot data candidates with higher data accesses and the data blocks related to the tasks with longer execution delays. In this paper, we take the number of data accesses as the *first priority* when extra data blocks are considered, and take the execution delay of corresponding tasks as the *second priority* when two data blocks have the same accesses. Note that, when we consider the second priority, those pending data blocks have already been used and their accesses have already been larger than 1.

Therefore, the completion of those completed tasks can be used as the guidance for choosing the data blocks related to computing-intensive tasks. For those tasks assigned to the datacenter within the extra time lag, related variables should be changed, i.e., in line 5 of Algorithm 3.

The complexity of runData is $\mathcal{O}(\Upsilon + \max_j|\boldsymbol{\mathcal{T}_j}|)$ for each submitted job, where Υ is the cost for solving a linear program over at most $\max_j|\boldsymbol{\mathcal{T}_j}|$ concurrent tasks. That is, Algorithm 3 contains the solving part for lp-GeoTS. Given the number of slots, the cost of the linear programming grows with respect to the increasing number of concurrent tasks. By using interior point method, the complexity is $\Upsilon = \mathcal{O}\big((\max_j|\boldsymbol{\mathcal{T}_j}|)^{\varpi} L\big)$ [56] in

worst case, where $\varpi \leq 3$ and L is the number of input bits. And **Algorithm 3** offloads at most $\max_j|\mathcal{T}_j|$ tasks.

5 Theoretical Analysis

Roadmap. In order to analyze the overall completion time for a sequence of jobs, we need to analyze the completion time of each job first. Therefore, we illustrate the theoretical analysis on runData by two key steps: **1)** Section 5.1 is used to bound the completion time for each job through Martingale Analysis; **2)** Section 5.2 is used to illustrate the relationship between the overall completion time for a sequence of jobs and its global optimum, which is our main theorem. The details of proofs are shown in Section 5.3.

5.1 Analysis on LocalOpt-Based Offloading

The analysis on the completion time of a single job contains three parts: the first Lemma analyzes those tasks assigned at edges; the second Lemma analyzes those tasks offloaded to the datacenter; and finally the third Lemma analyzes the overall completion time of a job. Here, for simplicity, we define the constants: $\forall j, H_j \triangleq \max_{i,s}\{|\mathcal{R}_{js}|, |\mathcal{N}_{ji}|\}$.

Lemma 1 Analysis on Local Tasks. *The maximal completion time of local task is concentrated on its LocalOpt, i.e., $Opt(\tilde{C}_j)$, with high probability. More specifically, the following inequality holds with the probability of at least $1-\delta$:*

$$\forall j:\ Local_j \leq Opt(\tilde{C}_j) + \max_{k\in\mathcal{T}_j, i\in\mathcal{L}_k}\{\gamma_i e_k\}\sqrt{2H_j \ln\frac{1}{\delta}}. \tag{1}$$

Proof. See Section 5.3.1, "Proof for Lemma 1". □

Lemma 2 Analysis on Remote Tasks. *The maximal completion time of remote tasks is also concentrated on its LocalOpt, i.e., $Opt(\tilde{C}_j)$, with high probability. More specifically, the following inequality holds with the probability of at least $1-\delta$:*

$$\forall j:\ Remote_j \leq Opt(\tilde{C}_j) + \max_{k\in\mathcal{T}_j, i\in\mathcal{L}_k}\{\sigma_i\}\sqrt{2H_j \ln\frac{1}{\delta}}. \tag{2}$$

Proof. See Section 5.3.2, "Proof for Lemma 2". □

Remark. We should mention here that Lemma 2 further implies the violation of Constraint (0) in εD-GeoTS, since the previous inequality holds under a certain probability, i.e., with the probability of at least $1-\delta$. However, since $Remote_j$ is concentrated on the local optimum of job-j, the violation is actually measured by the second term on the right and ε_j with the probability of at least $1-\delta$.

Lemma 3 Analysis on Entire Job. *The overall latency of job-j by using the power of two choices is concentrated on its LocalOpt with high probability. More specifically, the following inequality holds with the probability of at least $1-\delta$:*

$$\forall j:\ \Omega_j \leq Opt(\tilde{C}_j) + \max_{k\in\mathcal{T}_j, i\in\mathcal{L}_k}\{\sigma_i, \gamma_i e_k\}\sqrt{2H_j \ln\frac{1}{\sqrt{\delta}}}. \tag{3}$$

Proof. See Subsection 5.3.3, "Proof for Lemma 3". □

5.2 Analysis on A Sequence of Jobs

Theorem 1 Theoretical Analysis on A Sequence of Jobs. *The overall latency for n jobs is concentrated on its GlobalOpt with high probability by using runData. More specifically, following inequality holds with the probability of at least $1-n\delta$:*

$$\sum_j C_j \leq GlobalOpt + \mathcal{O}(\Psi_{\delta,n}),$$

$$\Psi_{\delta,n} \triangleq \kappa n \max_i \sigma_i + \max_j\left\{\varepsilon_j, \max_{k,i}\{\sigma_i, \gamma_i e_k\}\sqrt{2H_j \ln\frac{1}{\sqrt{\delta}}}\right\}, \tag{4}$$

where $\Psi_{\delta,n}$ describes a profile of the cloud-edge system, and κ is the average number of data accesses for a sequence of jobs.

Proof. See Section 5.3.4, "Proof for Theorem 1". □

We should mention here that: **1)** Although the probability in the main theorem contains the term of δ, the value of the term $\ln\frac{1}{\sqrt{\delta}}$ is acceptable in realistic settings. Here, we take 1000 jobs as a simple example. If the desired probability is 0.9, then δ equals 0.0001, and $\ln\frac{1}{\sqrt{\delta}}$ equals 4.6; **2)** κ and $\max_j H_j$ in the theorem illustrate the peak workload and the average workload for a sequence of jobs, respectively. As illustrated in the trace of Microsoft Cosmos, the average number of data accesses is 3.2, although the whole datasets are accessed in an unbalanced manner; **3)** Although $\Psi_{\delta,n}$ may be large in terms of a large job number n, in the tight version of the theorem, i.e., in Section 5.3.4, $\Psi_{\delta,n}$ actually shows the maximal distance between the result of runData and the lower bound of the global optimum. As a result, $\Psi_{\delta,n}$ just decides a range, instead of the exact performance.

5.3 Details on Proofs

5.3.1 Proof for Lemma 1

Proof. Given task-k with required data block stored in edge-i, a variable Δ_s^k is defined as $\gamma_i\sum_{x\prec k}(I_{xs}-p_{xs})\cdot e_x$, $\forall s\in\mathcal{S}_i$, where $\gamma_i\sum_{x\prec k} I_{xs}e_x$ describes the workload incurred on slot-s by all considered tasks. Here, the consider tasks are those whose submission timestamps are smaller than task-k (in other jobs) or task IDs are smaller than task-k (in the same job). Then, $\gamma_i\sum_{x\prec k} p_{xs}e_x$ describes the expectation of such workload on slot-s, since $\forall k, \mathbb{E}[I_{ks}] = p_{ks}$. Thus, given task-$k$, we have $\mathbb{E}[\Delta_s^k] = 0$, and we also have

$$\forall s,k: |\Delta_s^{f(k)} - \Delta_s^k| = \gamma_i|(I_{f(k)s} - p_{f(k)s})\cdot e_{f(k)}| \leq \gamma_i e_{f(k)},$$

where $f(k)$ is the successor of task-k in terms of the submission timestamp and task ID over considered tasks. And the partial inequality is defined based on the standard just mentioned. Since $\forall s\in\mathcal{S}_i$, $\{\Delta_s^k\}$ is a Martingale sequence [57]. Applying the Azuma's Inequality [58] on $\{\Delta_s^k\}$, we have

$$\forall s: Pr[\Delta_s^{g(s)} \geq \delta] \leq exp\left\{-\frac{\delta^2}{2\sum_{k\preceq g(s)}(\gamma_i e_k)^2}\right\}, \tag{5}$$

where $g(s)$ denotes the "maximal" task ID over considered tasks already assigned on slot-s, i.e., those queued tasks on slot-s based on the partial order exactly mentioned before. Inequality (5) equals that the following inequality holds $\forall s$, with the probability of at least $1-\delta$:

$$\begin{aligned}\sum_{k\preceq g(s)}(I_{ks}-p_{ks})e_k &\leq \max_{k\preceq g(s)}\{e_k\}\sqrt{2\max_{s,I_{ks}=1}\{|\mathcal{R}_{js}|\}\ln\frac{1}{\delta}}\\ &\leq \max_k\{e_k\}\sqrt{2H_j\ln\frac{1}{\delta}}.\end{aligned} \tag{6}$$

Since the slot with the maximal completion time among the edges also holds the previous inequality, we suppose that task-k is the straggliest task on slot-s without loss of generality. As a result, we have the equation regarding the straggliest task of job-j as follows:

$$\forall j: Local_j = q_{js}+\gamma_i\sum_{k'\in\mathcal{T}_j,s\in\mathcal{S}_i,i\in\mathcal{L}_k} I_{k's}e_{k'},$$

as well as the fact that the following inequality holds with the probability of at least $1-\delta$, i.e., $\forall j$:

$$\begin{aligned}Local_j &\leq \gamma_i\sum_{k'\preceq g(s)}p_{k's}e_{k'}+\gamma_i\max_{k'}\{e_{k'}\}\sqrt{2H_j\ln\frac{1}{\delta}}\\ &\leq Opt(\tilde{C}_j)+\max_{k'\in\mathcal{T}_j,i\in\mathcal{L}_{k'}}\{\gamma_i e_{k'}\}\sqrt{2H_j\ln\frac{1}{\delta}},\end{aligned}$$

where $Opt(\tilde{C}_j)$ is larger than the maximal completion time regarding the tasks on slots, as defined in lp-GeoTS. □

5.3.2 Proof for Lemma 2

Proof. By using the similar technique in Lemma 1, given task-k with required data stored in edge-i, Δ_k is defined as

$$\forall k: \Delta_k \triangleq \sigma_i\sum_{y\prec k}(X_y-\tilde{X}_y)+e_k\cdot(X_k-\tilde{X}_k).$$

Similarly, we have $\mathbb{E}[\Delta_k]=0$ and $|\Delta_{f(k)}-\Delta_k|\leq\sigma_i$. As a result, $\{\Delta_k\}$ is also a Martingale sequence. By applying Azuma's Inequality on $\{\Delta_k\}$, assuming task-k is the straggliest one as well as $|\mathcal{N}_i|=\sum_{k'\preceq h(i)}X_{k'}$ where $h(i)$ denotes the "maximal" task ID among those tasks offloaded from edge-i to the datacenter, $\forall j,k$, we have

$$Remote_j \leq \sigma_i\sum_{k'\preceq h(i)}\tilde{X}_{k'}+\tilde{X}_k e_k+\sigma_i\sqrt{2|\mathcal{N}_{ji}|\ln\frac{1}{\delta}}. \tag{7}$$

Since $|\mathcal{N}_{ji}|\leq H_j$ upon the definition and the value of $Opt(\tilde{C}_j)$ is the maximal completion time of task in lp-GeoTS, $Opt(\tilde{C}_j)$ is actually larger or equal to the first two terms on the right, and then we complete the proof.□

5.3.3 Proof for Lemma 3

Proof. After combining Lemmas 1 and 2 together, the following inequality holds with the probability of at least $1-\delta$:

$$\forall j: C_j \leq Opt(\tilde{C}_j)+\max_{k\in\mathcal{T}_j,i\in\mathcal{L}_k}\{\sigma_i,\gamma_i e_k\}\sqrt{2H_j\ln\frac{1}{\delta}}. \tag{8}$$

By using the power of two choices mentioned in Algorithm 2, the probability of the event in which both of these two choices breaks previous inequality is at most δ^2. Then, the event that Ω_j holds such inequality is at least $1-\delta^2$. Note that, according to the power of two choices in runData, i.e., lines 5 to 6 in Algorithm 2, we have

$$\forall j: \Omega_j = \min\{\{C_j\}_1,\{C_j\}_2\}.$$

Then, we substitute δ^2 for δ, and the following inequality holds with the probability of at least $1-\delta$:

$$\forall j: \Omega_j \leq Opt(\tilde{C}_j)+\max_{k\in\mathcal{T}_j,i\in\mathcal{L}_k}\{\sigma_i,\gamma_i e_k\}\sqrt{2H_j\ln\frac{1}{\sqrt{\delta}}},$$

whose form is the same as that mentioned in Lemma 3. Such form implies that the gap on the right grows slowly with the decrease of δ, as mentioned in theoretical analysis. □

5.3.4 Proof for Theorem 1

Proof. We denote by $\mathbf{\Phi}_{ji}$ the set of tasks belonging to job-j whose data are stored in edge-i after the submission of job-j. Due to the fact that any feasible solution of εD-GeoTS is also a feasible solution of GeoTS, the maximal task completion time for current job-j in εD-GeoTS is naturally larger than its optimum in GeoTS, as well as its theoretical lower bound solved from lp-GeoTS. Thus, we have

$$\forall j: Opt(\tilde{C}_j)\leq Opt(C_j)\leq\max_i\{\sigma_i\cdot|\mathbf{\Phi}_{ji}|\}+\max_k\{e_k\},$$

because transferring all of the tasks with data re-distribution regarding all data blocks to the datacenter is also one of the feasible solution in GeoTS. Therefore, its value is actually no doubt larger than the optimum of GeoTS. By applying the previous inequality to Inequality (3), defining

$$\forall j: \Xi_j=\max_{k,i}\{\sigma_i,\gamma_i e_k\}\sqrt{2H_j\ln\frac{1}{\sqrt{\delta}}},$$

and applying the Union Bound [59], we have the fact that the following inequality holds with the probability of at least $1-n\delta$. That is, for $\sum_j C_j$, we have

$$\begin{aligned}&\leq\sum_j\big\{\max_i\{\max_k\{e_k\}+\sigma_i\cdot|\mathbf{\Phi}_{ji}|\}+\max_j\{\varepsilon_j,\Xi_j\}\big\}\\ &\leq GlobalOpt+\sum_j\big\{\max_i\{\sigma_i\cdot|\mathbf{\Phi}_{ji}|\}+\max_j\{\varepsilon_j,\Xi_j\}\big\}.\end{aligned}$$

Since the overall number of data accesses is κn, the total number of transferred tasks is no more than κn. Here, we define κ as the average accesses for a sequence of jobs. That means we have the following inequality:

$$\sum_j\max_i\{|\mathbf{\Phi}_{ji}|\}\leq\kappa n.$$

The number of overall data accesses describes the whole pattern. And more precisely prediction on the transmission

of related data blocks directly shortens such theoretical gap. After using the definition of $\Psi_{\delta,n}$ as we mentioned in Section 5.2, we then complete the proof. Note that $\Psi_{\delta,n}$ describes a profile of the cloud-edge system. □

6 Implementation and Testbed

Currently, the distributed file system balances the data itself according to the volume while the data analytics framework reads/operates the data blocks in a batch manner, e.g., 64KB for each batch, instead of changing the data distribution. In our implementation, the Executor within each Spark Worker keeps in store already transferred data during the execution, and valid the data as a qualified data block to HDFS.

6.1 Implementation Upon Spark

To implement *run*Data, we override some components in Spark upon HDFS, as shown in Fig. 6. We should mention here that Spark and HDFS are typical frameworks used in data analytics. And in this paper, we take Spark and HDFS as the example to illustrate our implementation. Note that we also implement similar functionalities upon Hadoop and HDFS. Here, the Driver in Spark is used for scheduling the tasks in a job, the Worker is used for managing a server, and the Executors are used to host multiple running tasks in the Worker. Main functionalities are listed as follows:

Queue Tasks in Slots. Traditionally, the compute slot, i.e., the Executor in Spark, launches the tasks immediately when it is idle as well as it receives the requests from the Spark Driver. As a result, Hadoop and Spark do not support queuing tasks, although a branch of Yarn [47] has discussed such functionality. In this paper, we propose to postpone the scheduling of related tasks for the same purpose. Note that the scheduling of tasks is conducted by the TaskScheduler in Spark Driver. *run*Data tracks the status of tasks and assigns selected slot to a pending task as long as the selected slot is idle. Otherwise, *run*Data cheats the pending tasks again and again as if there is no available resource for task execution.

Re-Distribute Data Block via Piggybacking. During the task execution, as mentioned in previous section, data block is fetched batch by batch, and would be directly discarded after being received and used. In this paper, we reserve the space in terms of a data block with fixed size in the memory, as illustrated in Fig. 6. After splitting the transferred batch data for task execution based on pre-defined delimiters, we keep it in store in memory instead of reusing existing small buffer, in order to avoid the data discards. After the completion of a offloaded task with data re-distribution label, the Executor in Spark would trigger the Block Validator to valid the data stored in memory to a qualified data block.

Validate Data Block to HDFS. The data block stored in memory is first flushed to a specific directory of the local file system. Then, a CMD based thread is used by a script to delete the meta information of target data block in HDFS. After that, another CMD based thread is triggered to put the desired data blocks back into HDFS, so as to change the data re-distribution. Note that all of these things are conducted after the execution of a task, which has less impact on the task. After the completion of running tasks, the original data blocks would be cleaned by the balancer of HDFS itself soon.

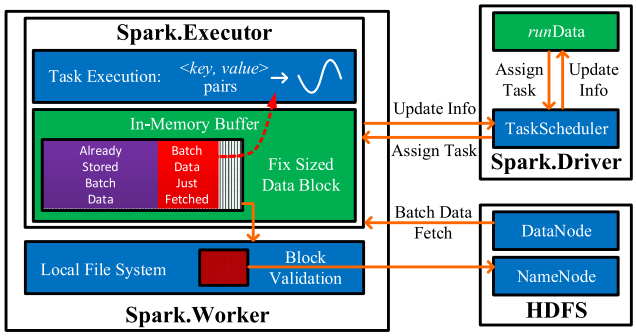


Fig. 6. Implementation overview upon Spark and HDFS.

After that, we illustrate our proposed *run*Data according to its workflow, overhead and security as follows:

Workflow of runData. As illustrated in Fig. 7a, the details of related workflow is shown. After receiving the *run*Data request along with the cluster message from Spark Driver, the executor then launches the task according to the *run*Data flag within the message, which indicates whether data re-distribution via piggybacking is needed. After receiving the batch data, *run*Data overrides the LineReader and TextInputFormat classes, in order to use those kept data in memory for task execution via $< key, value >$ pairs.

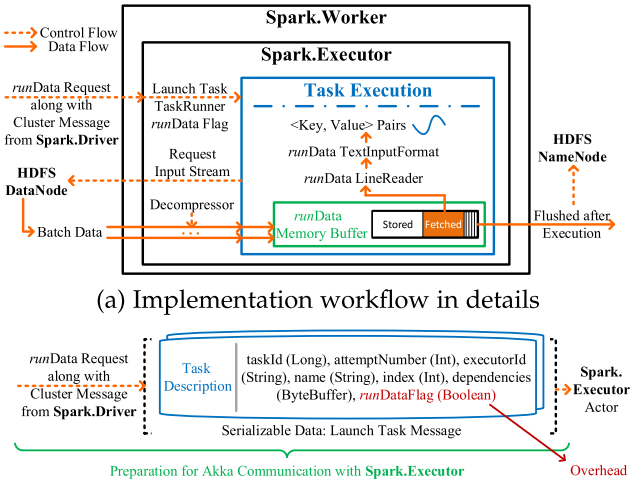
Overhead of runData. As shown in Fig. 7a, the operations related to store each batch data in the memory are actually implemented by just shifting a point, in order to distinguish those already stored data and those just fetched. Further shown in Fig. 7a, although the operations related to I/O flush are conducted, they are actually triggered after the execution of the task. As shown in Fig. 7b, the *run*Data message is actually a boolean flag to indicate whether the data re-distribution needs to be conducted.

Security of runData. Although the data re-distribution is actually conducted by validating flushed data block as if *run*Data cheats the NameNode of HDFS, all of the communications with NameNode incurred are conducted by using available APIs. As a result, the operations only triggered by those authorized servers are adopted, which are authorized during the initialization of HDFS by using authorized SSH keys and the check of HDFS itself. Thus, we assume that the authorized servers being checked are trustful.

Remark. Unfortunately, the available APIs supported by the HDFS fail to re-distribute the data blocks directly, since HDFS supports a logical disk for the users and the users do not need to consider the locations of related data blocks. As a result, we propose a mechanism to re-distribute the data blocks via piggybacking along with the execution.

6.2 Testbed Results

The testbed is built upon Spark 1.3.1, Hadoop 2.2.0, within 11 VMs deployed over four racks over Inspur SN5160M4, Dell PowerEdge R730 and R740, and Dell C6320, as shown in Fig. 8. One of the VM is responsible for both Spark Master (standalone mode) and HDFS NameNode. The default size of a data block in HDFS is 128MB. All of the VMs are used for both data storage and in-memory computation, whose settings range from 3 to 15 CPU cores, 8 to 20 GB memory and 15Mbps links by using Linux Traffic Control [60], [61]. The data randomly chosen from novels for WordCount is



(a) Implementation workflow in details

(b) Overhead for launching tasks

Fig. 7. Implementation details (workflow and message) over Spark.

1.2GB. Particularly, 15 CPU cores and 20GB memory are allocated to the master VM hosted in Inspur SN5150M4, to mimic the datacenter; The average CPU cores and memory allocated to Slave1 - Slave3 are 6 and 10GB, respectively, to mimic those edges with strong computing capability; And the CPU cores and the memory allocated to Slave4 - Slave10 are 3 and 8GB, respectively. All of the slaves connect to the master VM with authorized SSH keys mentioned before.

Algorithms. We compare proposed *run*Data with other four typical algorithms used in the testbed. **1)** *Locality* [9] runs tasks directly to the server, in which their required data is stored; **2)** *Aggregation* [13] aggregates all required data to the server hosting the scheduler for execution; **3)** *Delay* [62] assigns tasks to idle servers near to the related data blocks within acceptable time lag; **4)** *LocalOpt* schedules the tasks by using the results from *lp*-GeoTS as probabilities. Note that *Delay* scheduling is the default strategy in Spark.

Improvements. As shown in Table 2, *run*Data gains 15 percent reduction on average for a sequence of jobs. Initially, all of the data blocks are generated within the slaves randomly over four racks to mimic the scenario, where all of the data are often generated within edges. During the submission in term of a sequence of jobs with multiple trials, those hot data blocks are transferred by *run*Data and kept within the master as early as possible. For a majority of scenarios, all of the hot data blocks are transferred within the lifecycle of 10 jobs. Although Local*Opt* actually offloads the tasks to the master, it ignores to re-distribute the data blocks due to the limitation of those data analytics frameworks.

TABLE 2
Results (Seconds) Under Various Workloads Over Testbed

	Locality	Delay	Aggregation	Local*Opt*	*run*Data
High	50	47	59	35	32
Middle	35	37	58	30	26
low	35	24	37	23	18

For the scenarios, where the data blocks are generated and crowded within a few slaves, multiple accesses by a sequence of jobs easily result in heavy workloads. Although *Delay* Scheduling also offloads the tasks, it has to wait for several seconds and fails to re-distribute the data. *run*Data gains at least 27 percent reduction under low load scenario with 200Mbps bandwidth because *run*Data has much more opportunities for data re-distribution within extra time lag, i.e., $\varepsilon = 2$ according to the theorem. *run*data also gains at least 8.5 percent reduction under high load scenario. *Aggregation* assigns the tasks to the master, leading to long transmissions.

As shown in Fig. 9a, *run*Data only costs several million seconds for each batch data, including shifting the pointer in memory and other related operations, which nearly does no harm to the execution of a task. Here, the overall duration of a task under middle load is about 10 seconds. That means, the lag between two consecutive timestamps in Fig. 9a is about 0.1 second. As shown in Fig. 9b, *run*Data spends at most 150 million seconds to conduct *lp*-GeoTS.

7 Experimental Study

7.1 Methodology

The average job completion time is defined as the average elapsed durations for jobs, which start from jobs' arrival time and end at the moment that all their tasks completed.

Workloads. We use the synthetic workloads in our experiments with trace-driven distributions in terms of the job sizes, i.e., the number of tasks within a job, obtained from Facebook [16], [63] as well as Google production cluster [24], [64]. In order to make these two workloads consistent, we define the average percentage here regarding the occupied compute slots as the system utilization in our experiments.

Algorithms. Apart from four algorithms mentioned in the testbed, we further evaluate two more strategies commonly used: 1) *Frequency-Based* offloads hot data according to their accesses and a pre-defined threshold, e.g., 5; and 2) *Flutter* [15] offloads the tasks, considering the idle compute slots for optimized job completion time. Furthermore, for all of

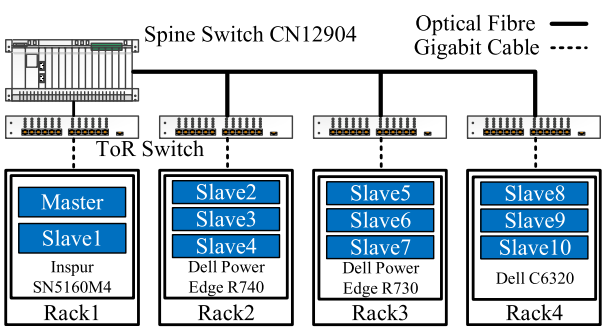
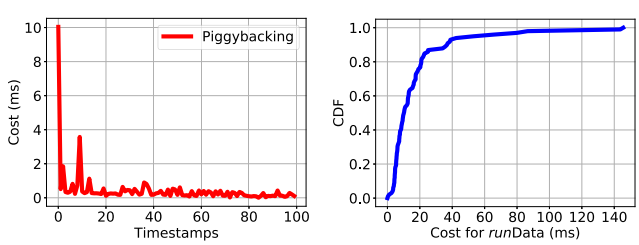


Fig. 8. Testbed topology deployed over four racks with 11 VMs.



(a) Cost for piggybacking (b) Cost for *run*Data

Fig. 9. Overheads of *run*Data with data re-distribution via piggybacking.

TABLE 3
Improvement by Data Re-Distribution via Piggybacking

Improvement via Piggybacking[1]	*Delay*$^+$ *v.s. Delay*	*Aggreg.*$^+$ *v.s. Aggreg.*
	54.99%	**68.16%**
	Flutter$^+$ *v.s. Flutter*	*LocalOpt*$^+$ *v.s. LocalOpt*
	55.98%	**25.41%**

1. The term $^+$ means data re-distribution via piggybacking is used.
2. The term Aggreg. is the abbreviation of Aggregation strategy.

the strategies, their extended versions with data re-distribution via piggybacking are also considered and evaluated. The notation used for the extended algorithm is $^+$.

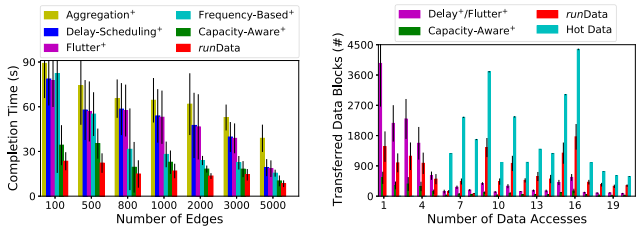
Setups. We adjust the number of edges ranging from 100 to 5000, in order to mimic various realistic scenarios [65]. The number of the compute slots and the WAN bandwidth are set based on the real performance analysis in terms of Amazon EC2 and previous studies [66], [67]. To evaluate the unbalanced data distribution, Zipf Distribution is used to model the skewness, which is adjusted by the Zipf's skew parameter [17]. The task durations are modeled by Pareto distribution with $\beta = 1.259$ based on the Facebook workload described in [26]. The jobs' inter-arrival timestamps are based on Poisson Process mentioned in previous work [16].

7.2 Experimental Results

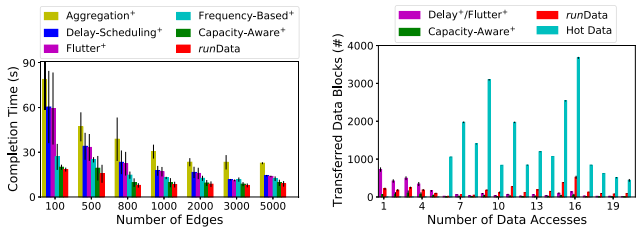
Except for the common metric, i.e., the average job completion time, we also evaluate the uploaded data.

Improvement of Piggybacking. Table 3 shows the reduction on the average job completion time for a sequence of jobs, by using the data re-distribution via piggybacking over multiple strategies. All of these four typical strategies with data re-distribution via piggybacking gain at least 25.41 percent improvement compared with their original versions, under the scenario, where the system is under middle load, i.e., 50 percent compute slots are occupied. Furthermore, other three strategies improve much more than that of Local*Opt*. The reason is that Local*Opt* has already taken the computing-intensive tasks as well as the resources at edges into consideration, and is willing to offload proper tasks to the datacenter. As a result, Local*Opt* has the lowest job completion time compared with other three strategies without data re-distribution. Hence, Local*Opt* has opportunities to transfer extra valuable data to the datacenter based on Local*Opt* itself. In overall, data re-distribution via piggybacking actually reduces the average job completion time.

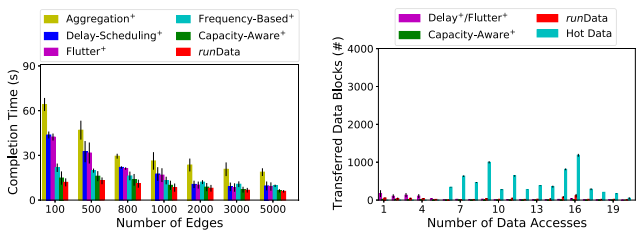
Characteristics of Uploaded Data. Except for the overall completion time for a sequence of jobs considered, we also analyze the uploaded data according to their accesses. Note that *Capacity-Aware* is the same as Local*Opt*, since both of them show the preference on the compute slots. As shown in Figs. 10a, 10b, and 10c, when the average system utilization is high, i.e., 80 percent, *run*Data uploads more hot data, which is transferred by *Frequency-Based* with data re-distribution via piggybacking. The results by using various threshold values for *Frequency-Based* are shown in Table 4. With the growth of threshold value, data blocks are more likely to be crowded within edges until their accesses reach to such threshold. As a result, the overall completion time increases correspondingly. *run*Data uploads nearly 40 percent data



(a) Results under high load, i.e., utilization=80%



(b) Results under middle load, i.e., utilization=50%



(c) Results under low load, i.e., utilization=20%

Fig. 10. Results to illustrate the effectiveness of *run*Data.

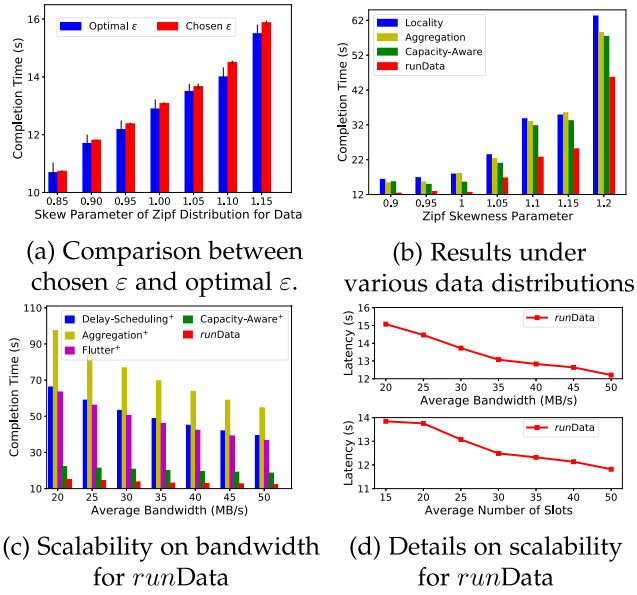
whose accesses are higher than 5. Although *run*Data uploads more data than *Flutter* and *Delay* Scheduling, most of the data transferred by them would not be frequently used by incoming jobs.

We should mention here that when the datacenter has adequate compute slots, *Flutter* uploads nearly the same volume of data compared with *Delay* Scheduling, because both of them prefer to assign tasks to idle slots. Moreover, nearly 90 percent data blocks are transferred via *run*Data when their accesses are less than 4, which means *run*Data actually offloads hot data candidates as early as possible by using data re-distribution via piggybacking and ε-bounded task scheduling. When the system utilization is low, *run*Data uploads fewer data blocks, but it has already tried its best to offload more hot data than other strategies. Furthermore, *run*Data uploads more data related to computing-intensive tasks to the datacenter, whose average completion time is at least 7 percent longer than the data transferred by other strategies.

Choices of ε. We also evaluate the choice of ε in order to show the effectiveness of the proposed theorem, as shown in Fig. 11a. Under the scenario where the utilization is moderate, i.e., 50 percent compute slots are occupied, the results of *run*Data regarding the average job completion time is very

TABLE 4
Results Under Various Frequencies
(Average Job Completion Time)

Threshold	0	1	2	3	4	5
Delay (s)	22.172	23.004	24.363	25.218	27.282	30.440
Threshold	6	7	8	9	10	11
Delay (s)	32.649	33.943	34.874	35.366	35.385	35.530



(a) Comparison between chosen ε and optimal ε.

(b) Results under various data distributions

(c) Scalability on bandwidth for runData

(d) Details on scalability for runData

Fig. 11. Results for validating the scalability of runData.

closer to the results by using optimal ε. The gap between these two results is at most 0.5s by using both estimated and optimal ε, under various scenarios with different skew parameters. ε actually relies on the skewness of data distribution, i.e., ε increases when the data distribution is much skewed within few edges. A large value of ε means it is necessary to re-distribute more data to the datacenter, but it would also defer current job while a small value of ε means runData has less opportunities on data re-distribution.

Scalability of runData. With the growth of Zipf skew parameter, as illustrated in Fig. 11b, those data blocks are more likely to be crowded within a few edges. Although the completion time of jobs increases, runData always performs the best. As illustrated in Fig. 11c, the average bandwidth ranges from 20MB/s to 50MB/s. With the increase of the bandwidth, data re-distribution via piggybacking has more opportunities to offload those hot data as early as possible. Thus, the gap between runData and other strategies with data re-distribution via piggybacking actually decreases, but also gain at least 9.1 percent reduction. *Aggregation*$^+$ decreases much when the bandwidth is high since the whole dataset is transferred, which is highly influenced by the bandwidth.

Both *Capacity-Aware*$^+$ and runData decrease less than that of any other strategies because they would also queue adequate tasks within the edges for optimizing the completion time for current job. Similar results also shown in terms of the compute slots. As shown in Fig. 11d, the details regarding the results of runData are illustrated. With the increase of both bandwidth and the average number of slots, the average job completion time decreases accordingly.

Table 5 shows the details of the traces used in large scale of evaluations. More specifically, 96 percent jobs generated in the trace of Google have 1-150 tasks while 8 percent jobs generated in the trace of Facebook have 150-500 tasks. In order to make these two workloads consistent, as mentioned before, we use the synthetic workloads with these two traces together such that the average percentage of occupied compute slots is middle for evaluations, i.e., the system utilization is 50 percent.

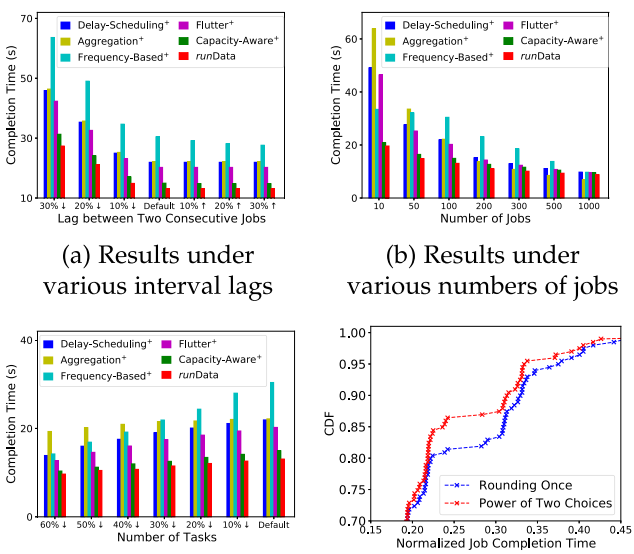
TABLE 5
Traces Used in Experiments

Traces Derived from Clusters	Distribution of Tasks (Ratio)		
	1-150 Tasks	150-500 Tasks	>500 Tasks
Google [24], [64]	96%	2%	2%
Facebook [16], [63]	89%	8%	3%

As illustrated in Fig. 12a regarding further results on the scalability, we vary the values of interval lags for two consecutive jobs. With the decrease of such interval lag for two consecutive jobs, more tasks are more likely to be crowded within the edges for a short time. runData balances proper tasks queued at the edges and those tasks transferred to the datacenter with data re-distribution. As a result, the reduction on the average completion time increases.

As illustrated in Figs. 12b and 12c, we vary both the number of jobs and the number of tasks in a job for evaluations. When the number of jobs is large, *Aggregation*$^+$ performs better than runData in terms of the average job completion time, since almost all data blocks are transferred to the datacenter and the forthcoming jobs are benefit from those transferred data blocks. However, the completion time for those jobs submitted at the very beginning is extremely large, which is unsuitable for timely data analytics. runData, instead, re-distributes the data along with the execution of tasks as well as along with the execution of jobs. Then, for any job submitted, the completion time is almost acceptable. With the decrease of the number of tasks, the reduction on the job completion time incurred by runData also decreases, since the re-distribution of data is conducted along with the tasks. Then, the decreasing number of tasks refers to less opportunity to re-distribute the data blocks.

At last, Fig. 12d shows the effect of the power of two choices used in Algorithm 2, which is the illustration of CDF. By using the power of two choices, the bad event in



(a) Results under various interval lags

(b) Results under various numbers of jobs

(c) Results under various numbers of tasks

(d) Comparison between rounding once and twice

Fig. 12. Further results on the scalability of runData.

terms of the long completion time for each job is more likely to be avoided. As a result, given fixed job completion time, the proportion of jobs by using the power of two choices is large than that of rounding the results from *lp*-GeoTS once.

8 Conclusion

Due to the throwaway feature of the data fetcher in data analytics frameworks, and multiple accessed datasets generated within edges, re-distributing the data via piggybacking would benefit forthcoming jobs. To further pursue the global optimum for a sequence of jobs as well as to guarantee the completion time for current job and WAN usage, we design an online data-driven mechanism *run*Data, which offloads the tasks with probabilities and can be proved concentrated on its optimum. We implement *run*Data upon Spark and HDFS and deploy our prototype within 11 VMs over 4 racks. The results show that *run*Data achieves 37 percent reduction compared with those state-of-the-art task scheduling schemas, covering a wide range of realistic settings.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under Grant 2017YFB1001801, in part by the National Science Foundation of China under Grants 61832005 and 61872175, in part by the Ripple Faculty Fellowship, in part by the Natural Science Foundation of Jiangsu Province under Grant BK20181252, in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization, in part by the Nanjing University Innovation and Creative Program for PhD under Grant CXCY19-25, in part by the Hong Kong RGC Research Impact Fund (RIF) through Projects R5060-19 and R5034-18, in part by the General Research Fund (GRF) through Projects 152221/19E and 15220320/20E, in part by the Collaborative Research Fund (CRF) through Project C5026-18G, in part by the National Natural Science Foundation of China under Grant 61872310, and in part by the Shenzhen Science and Technology Innovation Commission under Grant R2020A045. The preliminary version of this work entitled "Run Data Run! Re-distributing Data via Piggybacking for Geo-distributed Data Analytics" was presented in part at IEEE ISPA 2019 [1].

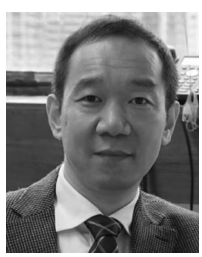
References

[1] Y. Li *et al.*, "Run data run! Re-distributing data via piggybacking for geo-distributed data analytics," in *Proc. IEEE Int. Sump. Parallel Distrib. Process. Appl.*, 2019, pp. 356–363.
[2] Google Datacenters, 2020. [Online]. Available: http://www.google.com/about/datacenters/
[3] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the expansion of Google's serving infrastructure," in *Proc. ACM IMC Conf. Int. Meas. Conf.*, 2013, pp. 313–326.
[4] Google Company, 2020. [Online]. Available: http://www.google.com/
[5] Microsoft Datacenters: Azure, 2020. [Online]. Available: https://azure.microsoft.com/en-us/global-infrastructure/
[6] Alibaba Cloud Computing, 2020. [Online]. Available: http://www.aliyun.com/
[7] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers," *Comput. Netw.*, vol. 130, pp. 94–120, 2018.
[8] W. Ouyang *et al.*, "Deep spatio-temporal neural networks for click-through rate prediction," in *Proc. ACM Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 2078–2086.
[9] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 323–336.
[10] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 377–392.
[11] Google AdWords, 2020. [Online]. Available: https://adwords.google.com/
[12] Q. Pu *et al.*, "Low latency geo-distributed data analytics," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 421–434.
[13] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and degradation in jetstream: Streaming analytics in the wide area," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2014, pp. 275–288.
[14] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 407–420.
[15] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling tasks closer to data across geo-distributed datacenters," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[16] C. C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proc. ACM Symp. Cloud Comput.*, 2015, pp. 111–124.
[17] Y. Jin, Z. Qian, S. Guo, S. Zhang, X. Wang, and S. Lu, "Ran-GJS: Orchestrating data analytics for heterogeneous geo-distributed edges," in *Proc. ACM Int. Conf. Parallel Process.*, 2018, pp. 1–10.
[18] C. C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang, "Wide-area analytics with multiple resources," in *Proc. ACM EuroSys Conf.*, 2018, pp. 1–16.
[19] Facebook Workloads, 2014. [Online]. Available: https://github.com/SWIMProjectUCB/S WIM/
[20] G. Ananthanarayanan *et al.*, "Scarlett: coping with skewed content popularity in MapReduce clusters," in *Proc. ACM Conf. Comput. Syst.*, 2011, pp. 287–300.
[21] Apache Spark, 2020. [Online]. Available: http://spark.apache.org
[22] Netflix Outage, 2018. [Online]. Available: https://istheservicedown.com/problems/centu rylink/
[23] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "Graphene: Packing and dependency-aware scheduling for data-parallel clusters," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 81–97.
[24] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 185–198.
[25] G. Ananthanarayanan *et al.*, "Reining in the outliers in MAP-reduce clusters using mantri," in *Proc. USENIX Conf. Oper. Syst. Des. Implementation*, 2010, pp. 265–278.
[26] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu, "GRASS: Trimming stragglers in approximation analytics," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2014, pp. 289–302.
[27] J. Wang, J. Yin, J. Zhou, X. Zhang, and R. Wang, "DataNet: A data distribution-aware method for sub-dataset analysis on distributed file systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 504–513.
[28] E. Coppa and I. Finocchi, "On data skewness, stragglers, and MapReduce progress indicators," in *Proc. ACM Symp. CLOUD Comput.*, 2015, pp. 139–152.
[29] Y. C. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune in action: Mitigating skew in MapReduce applications," *VLDB Endowment*, vol. 5, no. 12, pp. 1934–1937, 2012.
[30] J. Chen *et al.*, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 919–933, Apr. 2017.
[31] F. Zhang, J. Cao, W. Tan, S. U. Khan, K. Li, and A. Y. Zomaya, "Evolutionary scheduling of dynamic multitasking workloads for big-data analytics in elastic cloud," *IEEE Trans. Emerg. Top. Comput.*, vol. 2, no. 3, pp. 338–351, Sep. 2014.
[32] L. D. Briceno *et al.*, "Heuristics for robust resource allocation of satellite weather data processing on a heterogeneous parallel system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 11, pp. 1780–1787, Nov. 2011.

[33] H. Karau and R. Warren, *High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark*. Sebastopol, CA, USA: O'Reilly Media, 2017.
[34] J. S. Rojas, Á. R. Gallón, and J. C. Corrales, "Personalized service degradation policies on OTT applications based on the consumption behavior of users," in *Proc. Int. Conf. Comput. Sci. Appl.*, 2018, pp. 543–557.
[35] B. Yu and J. Pan, "Location-aware associated data placement for Geo-distributed data-intensive applications," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 603–611.
[36] J. Chen, K. Li, Q. Deng, K. Li, and S. Y. Philip, "Distributed deep learning model for intelligent video surveillance systems with edge computing," 2019, *arXiv:1904.06400*.
[37] G. Ananthanarayanan, V. Bahl, L. Cox, A. Crown, S. Nogbahi, and Y. Shu, "Video analytics-killer app for edge computing," in *Proc. ACM Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2019, pp. 695–696.
[38] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1423–1431.
[39] H. Jeong, H. Lee, C. H. Shin, and S. Moon, "IONN: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 401–411.
[40] M. Li, Q. Zhang, and F. Liu, "Finedge: A dynamic cost-efficient edge resource management platform for NFV network," in *Proc. IEEE/ACM Int. Symp. Qual. Serv.*, 2020, pp. 1–10.
[41] L. Zhao, J. Wang, J. Liu, and N. Kato, "Optimal edge resource allocation in IOT-based smart cities," *IEEE Netw.*, vol. 33, no. 2, pp. 30–35, Mar. 2019.
[42] HDFS in Apache Hadoop, 2019. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
[43] Hadoop MapReduce, 2020. [Online]. Available: http://hadoop.apache.org
[44] Y. Jin *et al.*, "Provisioning edge inference as a service via online learning," in *Proc. IEEE Int. Conf. Sens., Commun., Netw.*, 2020, pp. 1–9.
[45] G. Ananthanarayanan *et al.*, "Real-time video analytics: The killer app for edge computing," *IEEE Comput.*, vol. 50, no. 10, pp. 58–67, Oct. 2017.
[46] K. Ousterhout, C. Canel, S. Ratnasamy, and S. Shenker, "Monotasks: Architecting for performance clarity in data analytics frameworks," in *Proc. ACM Symp. Oper. Syst. Princ.*, 2017, pp. 184–200.
[47] Queuing the Container Requests in the NM, 2017. [Online]. Available: https://issues. apache.org/jira/browse/YARN-2883
[48] T. Wang, Z. Qian, L. Jiao, X. Li, and S. Lu, "GeoClone: Online task replication and scheduling for geo-distributed analytics under uncertainties," in *Proc. IEEE/ACM Int. Symp. Qual. Serv.*, 2020, pp. 1–10.
[49] AWS Optimizing CPU Options, 2019. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-optimize-cpu.html
[50] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM Conf. SIGCOMM*, 2013, pp. 3–14.
[51] Y. Liu *et al.*, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4961–4971, Jun. 2020.
[52] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4854–4866, Jun. 2018.
[53] S. Jošilo and G. Dán, "Decentralized algorithm for randomized task allocation in fog computing systems," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 85–97, Feb. 2019.
[54] S. Jošilo and G. Dán, "Wireless and computing resource allocation for selfish computation offloading in edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2467–2475.
[55] T. Chen, Q. Ling, and G. B. Giannakis, "An online convex optimization approach to proactive network resource allocation," *IEEE Trans. Signal Process.*, vol. 65, no. 24, pp. 6350–6364, Dec. 2017.
[56] Interior Point Method, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Linear_ programming
[57] R. Mansuy and R. Sverdlove, "The origins of the word 'martingale'," *Electron. J. Hist. Probability Statist.*, vol. 5, no. 1, pp. 1–10, 2009.
[58] K. Azuma, "Weighted sums of certain dependent random variables," *Tohoku Math. J., Second Ser.*, vol. 19, no. 3, pp. 357–367, 1967.
[59] J. Katz Samuels, L. Jain, Z. Karnin, and K. Jamieson, "An empirical process approach to the union bound: Practical algorithms for combinatorial and linear bandits," 2020, *arXiv: 2006.11685*.
[60] Linux Traffic Control, 2001. [Online]. Available: http://lartc.org/manpages/tc.txt
[61] X. Gao, Z. Gu, Z. Li, H. Jamjoom, and C. Wang, "Houdini's escape: Breaking the resource rein of linux control groups," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2019, pp. 1073–1086.
[62] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.
[63] C. J. Wu *et al.*, "Machine learning at Facebook: Understanding inference at the edge," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 331–344.
[64] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. ACM Symp. Cloud Comput.*, 2012, pp. 1–13.
[65] Akamai Platform Architecture, 2017. [Online]. Available: https://www.netmanias.com /en/?m=view&id=oneshot&no= 5951
[66] D. Jiang, G. Pierre, and C. H. Chi, "EC2 performance analysis for resource provisioning of service-oriented applications," in *Proc. Serv.-Oriented Comput. Icsoc/Servicewave Workshops-Int. Workshops*, 2009, pp. 197–207.
[67] J. Schad, J. Dittrich, and J. A. Quian é Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *VLDB Endowment*, vol. 3, no. 1–2, pp. 460–471, 2010.

Yibo Jin (Student Member, IEEE) received the BS degree from the Department of Computer Science and Technology, Nanjing University, in 2017, where he is currently working toward the PhD degree under the supervision of professor Sanglu Lu. He was a visiting student with The Hong Kong Polytechnic University, Hong Kong, in 2017. His research interests include big data analytics and edge computing.

Zhuzhong Qian (Member, IEEE) received the PhD degree in computer science in 2007. He is currently a professor with the Department of Computer Science and Technology, and a member with the National Key Laboratory for Novel Software Technology, Nanjing University, China. He has authored or coauthored research papers in journals, such as *Transactions on Parallel and Distributed Systems*, *Transactions on Networking*, *Transactions on Computers*, and *Transactions on Mobile Computing*, and conferences, such as INFOCOM, ICDCS, SECON, and IPDPS. His research interests include cloud computing, edge computing, and distributed machine learning. He is the chief member of several national research projects on cloud computing and edge computing. He was the recipient of the best paper awards from IMIS 2013, ICA3PP 2014, and APNet 2018.



Song Guo (Fellow, IEEE) is currently a full professor with the Department of Computing, The Hong Kong Polytechnic University. He is also the Changjiang chair professor awarded by the Ministry of Education of China. He has authored or coauthored papers in top venues. His research interests include big data, edge AI, mobile computing, and distributed systems. He has been recognized as a highly cited researcher (Web of Science) and was the recipient of more than 12 best paper awards from IEEE/ACM conferences, journals, and technical committees. He is the editor-in-chief of the IEEE Open Journal of the Computer Society. He was on the IEEE Communications Society Board of Governors, IEEE Computer Society Fellow Evaluation Committee, and editorial board of a number of prestigious international journals, including *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, and *IEEE Internet of Things Journal*. He was also the chair of organizing and technical committees of many international conferences. He is an ACM distinguished member.

Sheng Zhang (Member, IEEE) received the BS and PhD degrees from Nanjing University, in 2008 and 2014, respectively. He is currently an associate professor with the Department of Computer Science and Technology, Nanjing University. He is also a member with the State Key Lab for Novel Software Technology. His research interests include cloud computing and edge computing. He has authored or coauthored more than 80 papers, including those in *IEEE Transactions on Mobile Computing, IEEE/ACM Transactions on Networking, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers*, MobiHoc, ICDCS, INFOCOM, SECON, IWQoS and ICPP. He was the recipient of the Best Paper Award of IEEE ICCCN 2020, the Best Paper Runner-Up Award of IEEE MASS 2012, and the 2015 ACM China Doctoral Dissertation Nomination Award. He is a senior member of the CCF.

Lei Jiao (Member, IEEE) received the PhD degree in computer science from the University of Göttingen, Germany. He is currently an assistant professor at the Department of Computer and Information Science, University of Oregon, USA. Previously he worked as a member of technical staff at Alcatel-Lucent/Nokia Bell Labs in Dublin, Ireland and also as a researcher at IBM Research in Beijing, China. He is interested in the mathematics of optimization, control, learning, and mechanism design, applied to computer and telecommunication systems, networks, and services. He publishes papers in journals such as *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Mobile Computing*, and *IEEE Journal on Selected Areas in Communications*, and in conferences such as INFOCOM, MOBIHOC, ICNP, and ICDCS. He is a recipient of the NSF CAREER Award. He also received the Best Paper Awards of IEEE LANMAN 2013 and IEEE CNS 2019, and the 2016 Alcatel-Lucent Bell Labs UK and Ireland Recognition Award. He served as a guest editor for IEEE JSAC. He was on the program committees of many conferences including INFOCOM, MOBIHOC, ICDCS, and IWQoS, and was also the program chair of multiple workshops with INFOCOM and ICDCS.

Sanglu Lu (Member, IEEE) received the BS, MS, and PhD degrees in computer science from Nanjing University, in 1992, 1995, and 1997, respectively. She is currently a professor with the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. She has authored or coauthored more than 80 papers in refereed journals and conferences. Her research interests include distributed computing, wireless networks, and pervasive computing.