# Optimizing Geo-distributed Data Analytics with Coordinated Task Scheduling and Routing

Laiping Zhao Yanan Yang Ali Munir Alex X. Liu Yue Li Wenyu Qu

**Abstract**—Recent trends show that cloud computing is growing to span more and more globally distributed datacenters. For geo-distributed datacenters, there is an increasingly need for scheduling algorithms to place tasks across datacenters, by jointly considering WAN traffic and computation. This scheduling must deal with situations such as wide-area distributed data, data sharing, WAN bandwidth costs and datacenter capacity limits, while also minimizing makespan. However, this scheduling problem is NP-hard. We propose a new resource allocation algorithm called HPS+, an extension to Hypergraph Partition-based Scheduling. HPS+ models the combined task-data dependencies and data-datacenter dependencies as an augmented hypergraph, and adopts an improved hypergraph partition technique to minimize WAN traffic. It further uses a coordination mechanism to allocate network resources closely following the guidelines of task requirements, for minimizing the makespan. Evaluation across the real China-Astronomy-Cloud model and Google datacenter model show that HPS+ saves the amount of data transfers by upto 53% and reduces the makespan by 39% compared to existing algorithms.

**Index Terms**—Data Analytics, Task Scheduling, Routing, Geo-distributed Cloud

## 1 Introduction

Emerging cloud applications (such as artificial intelligence and data analytics) have many distributed components, spread across many different locations, that work together to achieve application goals. The distributed components of these cloud applications often have diverse set of resource requirements (such as CPU, RAM and network bandwidth), which demand customized datacenter architectures. For example, in scientific fields such as astronomy, telescopes generate a lot of data that needs to be stored and processed to extract useful information from the telescope data, making these applications both data-intensive and compute-intensive respectively. [1]. To efficiently store and quickly process these large volumes of data, datacenters are built close to the telescopes, forming a wide-area distributed datacenter network, as these telescopes are distributed over a large area. To this end, one such effort has been made by the United States National Virtual Observatory and China Virtual Observatory (China-VO) to provide cloud computing resources for processing astronomical applications [2]. Similarly, many online application service providers (such as Google, facebook and Amazon) operate hundreds of thousands of servers in multiple geographic locations, to provide a variety of cloud services to their users [3].

The diverse resource requirements of cloud applications and the geo-distributed datacenter architecture of cloud systems raises several challenges for task scheduling and routing in wide-area distributed datacenter networks.

For cloud application resource requirements, there are four key challenges in task scheduling and routing. Firstly, the task scheduling and routing is challenging because it is extremely expensive to process or transfer *large data volumes* across multiple datacenters [4], [5]. The explosive data growth is driving datacenters to manage data storage scaling from terabyte (TB) to petabyte (PB). For example, facebook currently handles 250PB of data in their warehouse generated by thousands of machines across multiple global regions and this data is increasing at a rate of 600TB per day [6]. Besides, Large Synoptic Survey Telescope (LSST) of the United States generates a new image every 15 seconds, leading to a 30TB raw image per night [7]. Second, the data is often derived from *geographically dispersed sources* that include users, devices, and sensors located around the globe. The distance between tasks and their input data or the distance among input data sets, makes data routing more complex. Third, many tasks of the same applications often have *shared input data*. Moreover, a data set could be accessed by multiple tasks simultaneously. Therefore, the shared nature of the datasets further complicates the task scheduling and routing. Fourth, *multiple data replicas* are stored for the same data over multiple datacenters to achieve higher reliability. This complicates task scheduling because during the data processing, applications need to select one of the replicas as the input for the task.

For the geo-distributed cloud system, the task scheduling and routing is complex and challenging because the task scheduler has to not only deal with wide-area-network (WAN) bandwidth cost, but also with the datacenter storage and computing capability limits as well. Furthermore, the limited WAN bandwidth makes the wide-area data transfer very expensive [5]. *Geo-distributed big data analytics* can provide fast and efficient data processing by providing compute nodes close to the actual datasets, thus reducing the network cost and latency. In such an architecture, for an application consisting of a large number of tasks, each

• *L. Zhao, Y. Yang, Y. Li, W. Qu are with College of Intelligence and Computing, Tianjin University, China. Email: {laiping, ynyang, liyue_, wenyu.qu}@tju.edu.cn. A. Munir and A. X. Liu are with Department of Computer Science, Michigan State University, East Lansing, USA. Email: munirali@msu.edu, alexliu@cse.msu.edu.*

task is preferred to be scheduled on the datacenter storing its input data, namely *datacenter-level locality*. However, the limited computational capability of a datacenter cannot meet the excessive demand of the increasing number of tasks, especially when they are required to deliver full-datacenter-level data locality. On the contrary, if multiple tasks that share the same data sets are not assigned to the same datacenter, these data sets might have to be transferred multiple times across datacenters, leading to a large amount of WAN traffic and hence higher network costs.

In this work, we design a task scheduling and routing system called HPS+ for geographically distributed cloud systems with a goal to minimize the makespan (i.e., task completion time). The makespan of a task in large geo-distributed systems depends on the data transfer time in (or across) the network and the task execution time at the compute node. If an application comprises a large number of tasks, their makespan can vary considerably depending on the data transfer time and the task execution time. Moreover, due to the parallel nature of the tasks, the makespan of a job is equal to the completion time of its slowest task. Therefore, in order to minimize the makespan, the scheduler in HPS+ considers all tasks of a job rather than individual tasks in isolation.

HPS+ considers an integrated approach for jointly scheduling the three major types of cloud resources (i.e., computation, data and network), compared to existing approaches that assume the computation process is agnostic to network resources allocation. Therefore, to achieve its goals, HPS+ first models the combined task-data dependencies and data-datacenter dependencies as an augmented hypergraph, and adopts a hypergraph partition approach, for minimizing the WAN data transfer volume. It outperforms the previous approach through a single-phase hypergraph partition method, that is, distributing tasks according to datacenters' capacities within a single partition step efficiently. Since datacenters are heterogeneous, HPS+ is able to assign more tasks to the ones with larger capacity. Next, it considers sufficiently the risk of network contention among tasks, and designs an task-aware Routing and Bandwidth Allocation (RBA) algorithm to coordinate the data transfer stage and computation stage among tasks, for minimizing the makespan.

HPS+ faces various challenges in its design. The first challenge is how to do data replica selection. As the data files are replicated across multiple nodes, we need to choose which replicas to create. To address this challenge, one of the replicas is designated as the leader, and is responsible for the data transfer. Moreover, HPS+ groups tasks that share overlapped data to avoid duplicated data transfer, thus, saving network resources. The second challenge is how to place tasks in the network. Task placement must take into account data volume and data locality so that each task can analyze the required data from the best datacenter with higher locality. To address this challenge, HPS+ chooses the datacenter close to the input data with a goal to maximize the datacenter-level data locality and minimize the WAN data transfer. The third challenge is how to route flows in the network. Traffic flows on WAN are generated due to the non-local input data of tasks. Their endpoints depend on the selected data replica and task placement. We address this challenge by balancing the allocated bandwidth among data transfers to reduce the makespan. Tasks with longer computing stage are always prioritized for more bandwidth to reduce the data transfer time. Note that, prior art in this direction does not consider joint data and compute resource optimization.

We evaluate HPS+ performance using simulations. We submit tasks to two simulated geo-distributed cloud systems: the CVO cloud is configured from China-VO project [8] and consists of 5 datacenters, the Google cloud is configured from B4 network [9] and consists of 12 datacenters. We compare HPS+ with three other algorithms and find that HPS+ on average reduces the amount of data transfers by 45%-52% and obtains 33%-38% better makespan than existing mechanisms.

Our main contributions can be summarized as follows:

1. We propose a joint task scheduling and routing framework for geo-distributed cloud frameworks.
2. We model the combined task-data dependencies and data-datacenter dependencies as an augmented hypergraph, and present four techniques to improve the hypergraph partition approach, for minimizing the WAN data transfer volume. In a nutshell, these techniques include: (i) introducing unit-weighted "virtual tasks", (ii) negative weighted hypergraph nodes representing datacenters, (iii) double-counting free cut-cost function, (iv) zero-approaching balance constraint.
3. We also propose a Routing and Bandwidth Allocation (RBA) algorithm to minimize the makespan of the data transfer stage and computation stage.
4. We evaluate the proposed scheme using simulations and show that our proposed approach can reduce the WAN data transfer volume by upto 53%, and reduce the makespan by upto 39%.

The rest of the paper is organized as follows. Section 2 gives a scientific computing example that motivates our work. We present the system model to formulate our problem in section 3. Section 4 describes the design of task scheduling and routing algorithm. In section 5.1, we experimentally evaluate the performance of our proposal compared with other algorithms. Section 6 presents the related work. We conclude and discuss future work in section 7.

## 2 Motivation Example

### 2.1 Background

In this project, our goal is to provide astronomers with a cloud platform, which is able to realize the coordinated processing of astronomical data generated by telescopes at various locations. In this regard, we consider the cloud computing resources provided by the China Virtual Observatory (China-VO) for processing astronomical applications [2]. In astronomy, China has deployed a number of astronomical telescopes, including radio telescopes (such as the

Shanghai 65meter radio telescope), and optical telescopes (such as LAMOST in Xinglong and 2.4meter telescopes) in Lijiang [10]. These telescopes produce data for various parameters such as electromagnetic radiation, $\gamma$-ray, $X$-ray, infrared ray, etc., and need to extract useful information from them, which is a complex and time consuming task as it needs to process terabytes of data.

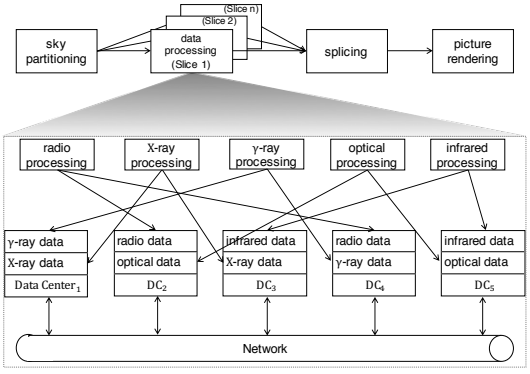


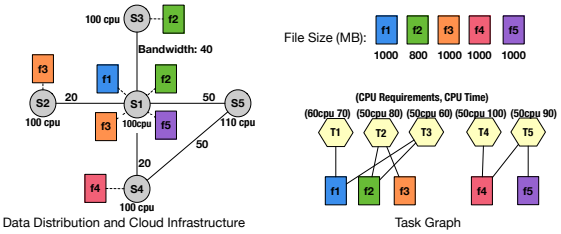

Fig. 1. An Overview of Visualization in Astronomy

Fig. 1 depicts the high level structure of an astronomical data visualization application [8], which has been studied in some well-known projects like VizieR Catalog service [11], WWT [12], Google Sky. In this application, the entire outer space is captured by joining the images captured in the form of a large number of small regions, according to their coordinates. First, each *data processing task* creates an image by processing multiple slices of data. Then, the images acquired from these tasks are further combined together and rendered to generate a panorama of space within different wavebands. For this purpose, each *data processing* task takes image data from multiple regions as input, and furthermore, each image data might be processed by multiple tasks due to the overlapping edges.

Our design goal is to minimize the makespan of these tasks, which is challenging because the data sets are widely dispersed over multiple distant datacenters.

## 2.2 Scheduling Results

Next, we compare the performance of different scheduling algorithms in terms of minimizing the makespan of different tasks. Fig. 2 summarizes the example settings (datacenters, WAN, task graph, and files distribution) based on the China-VO platform, and the scheduling results obtained by various scheduling techniques such as Greedy, CDS [13], HPS [8], Flutter [14] and our proposed HPS+. There are five datacenters configured with 100, 100, 100, 100 and 110 cores, respectively. Five network links with different available bandwidth connect the five datacenters, and five files with different size (1000, 800, 1000, 1000 and 1000 ) are replicated and distributed in datacenters. In particular, datacenter $S_1$ stores the files of $f_1$, $f_2$, $f_3$ and $f_5$, while $S_5$ stores nothing. The application consists of five tasks, processes the data and generates at most eight flows.




| | S1 | S2 | S3 | S4 | S5 | Data Transfer | Makespan (Exclusive BW) | Makespan (Shared BW) |
|---|---|---|---|---|---|---|---|---|
| Greedy | T1 | T2 | T3 | T4, T5 | | 2800 | 140 | 120 |
| CDS | T1 | T2 | T3 | T4, T5 | | 2800 | 140 | 120 |
| HPS | T3 | T2 | T1 | T4 | T5 | 3800 | 150 | 120 |
| Flutter | T2, T3 | | | T4 | T1, T5 | 3000 | 130 | 120 |
| HPS+ | T2, T3 | | T1 | T4, T5 | | 2000 | 140 | 104.3 |

Fig. 2. Example and Scheduling Results: Task Schedule, Data Transfer Volume and Makespan Computed by Different Scheduling Algorithms

### 2.2.1 Greedy Scheduling Algorithm (GS)

This algorithm greedily assigns tasks to the datacenter with maximum data locality. If the datacenter with maximum data locality cannot provide sufficient computing capacity, the task will be assigned to the second data-local datacenter, and so on. For our current example, task $T_1$ is assigned to datacenter $S_1$, $T_2$ is assigned to $S_2$, $T_3$ is assigned to $S_3$, $T_4$ and $T_5$ is assigned to $S_4$, generating WAN data transfer of 2800, and the makespan is 120.

### 2.2.2 Community Detection-based Scheduling (CDS)

By modeling the task scheduling as a community detection problem, this algorithm iteratively places the tasks into communities, while maximizing the modularity measure $Q = \frac{1}{2m}\sum_{i,j}[A_{ij} - \frac{k_i k_j}{2m}]\ \delta(c_i, c_j)$, where $A_{ij}$ is proportional to the data transfer volume [13]. In our current example, the $Q$-measure is consistent with the data locality measure. Hence, CDS produces the same scheduling results as the *GS*.

### 2.2.3 Hypergraph Partition-based Scheduling (HPS)

This algorithm assigns tasks to datacenters utilizing a hypergraph-partition method [8], which partitions the task graph into balanced groups whose number is equal to the number of datacenters. Balanced partitions leads to jagged completion time, due to the varieties of datacenters' capacities. Hence, *HPS* further reassigns tasks from the long-run datacenter to the short-run datacenter in order to balance the completion time, at the cost of increasing the total file transfer volume. Different with our model, they majorly focus on the situation that *the overall computation requirements* $\gg$ *the overall capacity supply*, and it allows tasks to wait in the queue if the datacenter does not have sufficient capacity to accomodate a group. *HPS* does not perform well in case of moderate computation requirements, since its objective is load balancing. In our example scenario, the five tasks are assigned to five different datacenters, generating WAN data transfer volume of 3800, and resulting in makespan of 120 in the example.

#### 2.2.4 *Flutter*

This algorithm formulates the scheduling problem as a lexicographical min-max integer linear programming (ILP) problem, and then transforms it into a nonlinear program with a separable convex objective function and a totally unimodular constraint matrix, which can be solved using a standard linear programming solver efficiently [14]. The original *Flutter* constrains the maximum number of tasks assigned to a datacenter instead of constraining the amount of requested resources, implying that all tasks require the same amount of resources. It also assumes that the network bandwidths are stable over time, meaning that the bandwidths of the shortest path are used exclusively by data flows. Hence, *Flutter* cannot directly solve our scheduling problem, where we consider the more realistic settings: each datacenter provides limited amount of resources; and network bandwidths available to applications varies significantly over time due to the sharing among data flows. We modify *Flutter* as follows: First, obtain the scheduling results using the original *Flutter*. Then, we adopt a rounding technique if its results is not an integer. Given the scheduling results, the makespan is derived using our bandwidth allocation algorithm with regard to shared network bandwidth. For the example scenario, although *Flutter* produces the earliest makespan (130) if using exclusive network bandwidth, its actual makespan in shared network environment is the same as *GS* and *CDS*, which is 120.

### 2.3 Shortcomings of the Scheduling Techniques

Both *GS* and *CDS* neglect the potential impact of task scheduling order on the WAN data transfer volume. *HPS* addresses this problem, but generates more WAN data traffic in order to achieve load balancing. *Flutter* further results in more WAN data transfer, since it assumes that it has sufficient bandwidth for WAN data transfer but in real network it may not actually have available bandwidth. To address these limitations of existing schedulers, we design HPS+ that avoids the load balancing limitation of *HPS* using and optimized algorithm to carefully arrange node weights and it also support precise bandwidth allocation in shared network. In summary, our HPS+ generates the least WAN data transfer (2000) and the earliest makespan (104.3) because it removes the load balance constraint indirectly and provides a coordination mechanism between task placement and flow routing in shared network environemt.

## 3 System Model

In this section, we describe the system model and the formulation of scheduling problem.

### 3.1 Overview

Fig. 3 presents the system architecture for task execution in the geo-distributed cloud. The system has two main components, a controller that schedules tasks in the network and resource monitors that collect network and application stats to help controller make scheduling decisions. The controller

TABLE 1
List of notations

| Notation | Definition |
|---|---|
| $G^S$ | $G^S = (N^S, E^S)$, the substrate cloud system. |
| $R_i$ | Capacity of datacenter $i$, $\forall i \in N^S$. |
| $R_{uv}$ | Capacity of link $uv$, $\forall uv \in E^S$. |
| $h_i$ | The required computation capacity by task $i$. |
| $G^V$ | $G^V = (N^V, E^V)$, tasks-files relations network: $N^V = N^T \cup N^F$ |
| $N^T$ | Task set. |
| $N^F$ | Data files set. |
| $E^V$ | Dependencies between tasks and data files. |
| $N_i^F$ | The data set processed by task $i$. |
| $N_f^T$ | The set of tasks taking file $f$ as input. |
| $N_f^S$ | The set of datacenters storing file $f$. |
| $s_f$ | Size of a data file $f$. |
| $p_i$ | CPU time of a task $i$. |
| $b_t^{fi}$ | Allocated bw on the substrate tunnel $t$ for flow $fi$. |
| $f^k$ | The $k$th replica of file $f$. |
| $N^*$ | The set of "virtual tasks". |
| $V$ | The set of vertices of hypergraph. |
| $E$ | The set of nets in hypergraph: $\|E\| = \|N^F\|$ |

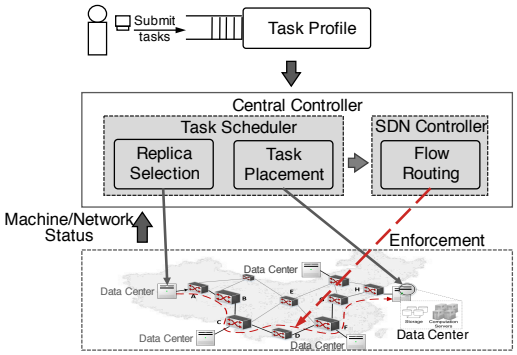



Fig. 3. System Architecture for Distributed Job Execution

consists of a task scheduler and a SDN controller. The task scheduler periodically starts the scheduling algorithm if there are jobs arriving. It assembles jobs as one "big job" and generates a schedule based on their requirements and datacenter state. Network flows fall into two categories: *non-interactive* and *interactive* flows. For *interactive* flows, the SDN controller predicts their demands in the next interval using a deep neural network model utilizing Long Short-Term Memory (LSTM) network [15], and reserves bandwidth for them. The flows generated by big data analytic jobs falls into *non-interactive* flows. Depending on the location of selected replica and task placement, the flow routing module computes the required bandwidth and makes the routing decision for each generated flow.

At each datacenter, the task and resource monitors manage the on-going tasks and collect application characteristics and machine resource usage status. It regularly sends "heartbeat" messages to the controller, reporting their latest changes. The controller obtains the network status (such as link bandwidth) from the openflow-enabled switches in the network and routes traffic accordingly.

To model the system, we denote the cloud system by a graph $G^S = (N^S, E^S)$, where $N^S$ and $E^S$ are the set of dat-

acenters and links between datacenters, respectively. Each datacenter provides heterogeneous storage and computation resources. Therefore, the available computational resource of a datacenter $i \in N^S$ is described as $R_i$, which refers to the available number of CPU cores. We assume that the data is already saved on some data center and we do not need to choose which datacenter we want to save each replica. Data is stored in file form in datacenters, where each data file can be replicated over multiple datacenters, and only one of replica is designated as the input for a task. The physical network among datacenters consists of entities such as routers, switches, links and hubs. The bandwidth capacity of a link $uv \in E^S$ is denoted by $R_{uv}$. Communications on the network follow the bi-directional multi-port model, which allows multiple incoming and outgoing communications simultaneously as long as the bandwidth constraints are not violated. We also assume that path splitting is supported by the substrate network.

### 3.2 Task Model

Big data analytics often divide the job processing into stages (like Mapreduce [16], Spark [17]), and a set of tasks in each stage can run in parallel on different machines. In our model, we only focus on the scheduling of tasks in a single stage, and the optimization of the end-to-end application performance is considered in our future work. Let $G^V = (N^V, E^V)$ be the task graph constructed by the set of tasks $N^T$ and their input files $N^F$, where $N^V = N^T \cup N^F$ and $E^V$ denotes the dependency relations between tasks and files. In our model, a file may be shared by multiple tasks, while a task may take multiple files as input. A task starts processing only after its input files are all transferred to local storage. Thus, the running time of a task is comprised of data transfer time and execution time. Suppose the file $f$ is one of the input for task $i$, and the size of a file $f$ is represented by $s_f$, then its data tranfer time can be computed as $s_f/b^{fi}$, where $b^{fi}$ denotes the allocated bandwidth to flow $fi$. If an input file for a task has already been transferred to local datacenter for some other task, the task will just read the local replica directly, and we do not transfer it again. Even in the same datacenter, it also takes time to transmit the data from the file server to the task server, although it may be an order of magnitude lower than that taken by WAN.

Each task is encapsulated in a VM and each VM is only allowed to run one task. Cloud providers advertise the capacity offered by multiple discrete types of VM instances and users make scheduling and placement decisions accordingly for VM instance to use for each task (denote by $h_i$ the required capacity by task $i$). We here only consider the case that cloud providers have sufficient capacities to accommodate all VMs, i.e., the starting times of all the tasks in a datacenter are the same. We estimate the CPU time (denoted by $p_i$) of a task through dividing its number of instructions by VM CPU speed (MIPS), that is, $p_i = \#of\ insts/VMspeed$, as the VM instance required for each task is chosen independently of the other load (users) in the network.

### 3.3 Problem Statement

Given a geo-distributed cloud system and task model described above, we seek a schedule to minimize the makespan, by orchestrating the replica selection, task placement and flow routing. We formulate the problem as a mixed integer programming (MIP) problem.

We define four variables to represent the sub-problems of replica selection, task placement and routing:

- $x_{ij}$: a binary variable, which has the value "1" if task $i$ is assigned to datacenter $j$. Otherwise, it is set to "0".
- $y_{f^k i}$: a binary variable, which has the value "1" if the $k$th replica of the input file $f$ of task $i$ is selected for data transfer. Otherwise, it is set to "0".
- $b_t^{f^k i}$: the bandwidth allocated on the substrate routing path $t$ for the data flow from a replica $f^k$ to task $i$.
- $z_t^{f^k i}$: a binary variable, which has the value "1" if $b_t^{f^k i} > 0$. Otherwise, $z_t^{f^k i} = 0$.

**Objective:**

$$minimize: \quad c \tag{1}$$

where $c$ is the makespan.

**Constraints:**

$$\frac{s_f}{b^{fi} + \varphi} + p_i \le c, \quad \forall f \in N^F, \forall i \in N^T \tag{2}$$

$$\sum_{fi \in E^V} \sum_{f^k \in \{f\}} \sum_{\{t | uv \in t\}} b_t^{f^k i} \le R_{uv}, \quad \forall uv \in E^S \tag{3}$$

$$\sum_{i \in N^T} h_i x_{ij} \le R_j, \quad \forall j \in N^S \tag{4}$$

$$\sum_{j \in N^S} x_{ij} = 1, \quad \forall i \in N^T \tag{5}$$

$$\sum_{f^k \in \{f\}} y_{f^k i} = 1, \quad \forall i \in N^T, \quad \forall f \in N_i^F \tag{6}$$

$$(y_{f^k i} - 1) z_t^{f^k i} = 0, \quad \forall t, f^k i \tag{7}$$

$$\sum_{f^k \in \{f\}} y_{f^k i} \sum_t z_t^{f^k i} > 0, \quad \forall fi \in E^V \tag{8}$$

$$(y_{f^k i} - 1) b_t^{f^k i} = 0, \quad \forall t, f^k i \tag{9}$$

$$\sum_{f^k \in \{f\}} y_{f^k i} \sum_t b_t^{f^k i} > 0, \quad \forall fi \in E^V \tag{10}$$

$$z_t^{f^k i} \left[ (x_{ij} I_{jt} - 1) + (y_{f^k i} I_{f^k t} - 1) \right] = 0, \quad \forall i, t, fi \tag{11}$$

$$\sum_{f^k \in \{f\}} y_{f^k i} \sum_j x_{ij} \sum_t I_{jt} I_{f^k t} z_t^{f^k i} > 0, \quad \forall fi \tag{12}$$

$$b_t^{f^k i} \left[ (x_{ij} I_{jt} - 1) + (y_{f^k i} I_{f^k t} - 1) \right] = 0, \quad \forall i, t, fi \tag{13}$$

$$\sum_{f^k \in \{f\}} y_{f^k i} \sum_j x_{ij} \sum_t I_{jt} I_{f^k t} b_t^{f^k i} > 0, \quad \forall fi \in E^V \tag{14}$$

$$b_t^{f^k i} + (1 - z_t^{f^k i}) > 0, \quad \forall f^k i, t \tag{15}$$

$$b_t^{f^k i} (1 - z_t^{f^k i}) = 0, \quad \forall f^k i, t \tag{16}$$

$$b_t^{f^k i} \ge 0, \quad \forall fi \in E^V, \forall f^k \in \{f\} \tag{17}$$

$$x_{ij}, y_{f^k i}, z_t^{f^k i} \in \{0, 1\}, \quad \forall ij, f^k i, t \tag{18}$$

- Constraint (2) enforces that all tasks are completed before $c$. Let $b^{fi} = \sum_t b_t^{f^k i}, \forall f \in N_i^F$. Since a task starts processing as long as its input data are transferred to local, we have $c_i = max_{f \in N_i^F}(s_f/(b^{fi} + \varphi)) + p_i$, $\forall i \in N^T$, where $\varphi$ is a small constant close to 0, and is used to avoid division by zero errors.
- Constraint (3) and (4) enforce the capacity bounds of the substrate links and datacenters.
- Constraint (5) makes sure that only one datacenter is selected for each task. Constraint (6) indicates that an input file for a task can only be transferred from a single replica.
- Constraint (7) together with (8) specify the correct correlation between $y_{f^k i}$ and $z_t^{f^k i}$. When $y_{f^k i} = 0$, $z_t^{f^k i}$ must also be 0 for ensuring (7). It means that, when $f^k$ is not selected as the input replica for task $i$, there exists no data transfer from $f^k$ to $i$. When $y_{f^k i} = 1$, constraint (7) is established. However, due to constraint 8, we conclude that there exists at least one routing path $t$ transmitting data from $f^k$ to $i$, when $f^k$ is designated as an input replica for task $i$. Likewise, constraint (9) and (10) together guarantee the correct correlation between $y_{f^k i}$ and $b_t^{f^k i}$:
- Constraint (11), (12) and (18) together make sure the correct correlation among $x_{ij}$, $y_{f^k i}$ and $z_t^{f^k i}$. In particular, $I_{jt}$ (or $I_{f^k t}$) has the value "1" if path $t$ uses $j$ ($f^k$) as the end (start) point and "0" otherwise. When $z_t^{f^k i} = 1$, we must have $x_{ij} I_{jt} = 1$ and $y_{f^k i} I_{f^k t} = 1$ simultaneously. It means that, when flow $f^k i$ is transmitted over path $t$, the start point of $t$ should be $f^k$ and the end point of $t$ should be $j$. On the other hand, when $z_t^{f^k i} = 0$, $x_{ij}$ and $y_{f^k i}$ could be either "0" or "1". Constraint (12) ensures that, when task $i$ is placed on $j$ and replica $f^k$ is selected for flow $fi$, there exists at least one tunnel $t$ for transmitting flow $fi$. Likewise, constraint (13), (14) and (17) together guarantee the correct correlation among $x_{ij}$, $y_{f^k i}$ and $b_t^{f^k i}$.
- Constraint (15) together with (16) specify the correct correlation between $b_t^{f^k i}$ and $z_t^{f^k i}$. Because of constraints (15), (17) and (18), when $b_t^{f^k i} = 0$, $z_t^{f^k i}$ must be 0 as well. Because of constraints (16), (17) and (18), when $b_t^{f^k i} > 0$, $z_t^{f^k i}$ must be 1.
- Constraint (17) and (18) refer to the domain constraints.

**Theorem 1.** *The problem of minimizing the makespan for data-intensive applications (Formula (1)-(18)) is NP-Hard.*

*Proof:* Let us first consider the decision problem corresponding to the above problem as follows: *Given a pre-defined makespan c, is there a scheduling and routing plan such that achieves a makespan of no largere than c while satisfying the constraints (2)-(18)?* It is easy to see that the problem is in NP: given a scheduling-routing plan, a positive answer to the decision problem is verifiable in time $O(n)$, where $n = |N^T|$.

Next, we show that this decision problem can be solved using the 3-*Partition* approach.

***3-Partition problem*** [18]: Given positive integers $\delta$, $\psi$, and a set of integers $X = \{x_1, x_2, ..., x_{3\delta}\}$ with $\sum_{k=1}^{3\delta} x_k = \delta\psi$ and $\psi/4 < x_k < \psi/2$, $\forall k$, does there exist a partition $\{\Omega_1, \Omega_2, ..., \Omega_\delta\}$ of $X$ with $|\Omega_l| = 3$ and $\sum_{x_i \in \Omega_l} x_i = \psi$, $\forall l \in [1..\delta]$?

Given the above definition of the 3-*Partition* problem, an instance of the minimizing makespan problem can be defined as follows: Suppose the number of tasks $n = 3\delta$, the number of datacenters $m > \delta$, and $h_i = x_i$, $\forall i \in [1..3\delta]$, so $\sum_{i=1}^{3\delta} h_i = \delta\psi$. Also assume that all the data files are replicated on a subset $\delta$ of all datacenters (denoted by $N^\delta$ the set of these $\delta$ datacenters), while the rest of datacenters do not store any data. Therefore, if a task is scheduled on one datacenter in $N^\delta$, it will not produce any network transmission since the data is available locally in that datacenter. However, if a task is scheduled on any one of the rest of datacenters, it starts processing only after network transmission (the data transfer time is greater than 0) is finished. For an arbitrary task $i$, we set $p_i = c$. Therefore, if any task is not scheduled to one datacenter in $N^\delta$, the objective function (Formula (1)) will be greater than $c$. That is, any task has to be scheduled on one of the datacenters in $N^\delta$. Now, if $R_i = \psi$, $\forall i \in N^\delta$, then $\sum_{i \in N^\delta} R_i = \delta\psi$. Since $\psi/4 < h_i < \psi/2$ for each task, this means we cannot schedule more than 3 tasks to any datacenter $i \in N^\delta$. Therefore, each datacenter will accomodate exactly 3 tasks because there are $3\delta$ tasks that need to be scheduled on $\delta$ nodes. Henceforth, a 3-Partition problem provides a solution to the decision problem. Conversely, a solution to the decision problem with $c = p_i$, $\forall i$ and $\sum_{i \in N^\delta} R_i = \delta\psi$ provides a solution to the 3-*Partition* problem. Since the problem with $c = p_i$, $\forall i$ and $\sum_{i \in N^\delta} R_i = \delta\psi$ is just a special example of the decision problem, it is at least as hard as 3-partition problem. □

## 4 Our Solution: HPS+

Since the underlying resource allocation problem is NP-Hard, large-scale instances and models make this problem computationally intractable. To avoid exploring all possible combinations of replica selection, task placement and flow routing, HPS+ adopts a joint algorithm that first orchestrates the replica selection and task placement for minimizing the volume of transferred data, i.e., maximizing the datacenter level-locality, and then uses a flow routing and bandwidth allocation mechanism to balance network load and minimize the completion time of each task.

### 4.1 Minimizing Data Transfer Volume

To minimize the volume of transferred data, each task is preferably assigned to the datacenter storing its input data, i.e., *datacenter-level locality*. However, the input data of a task may be dispersed over multiple sites, and the computation capacity of a datacenter is usually limited. Therefore, it is not always possible to accommodate all the tasks, satisfying full data-locality. Some tasks have to

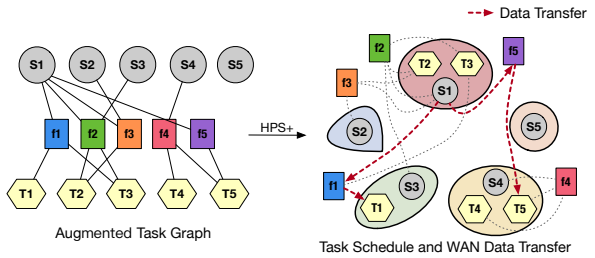



Fig. 4. Task scheduling: (1) Construct the augmented task graph, (2) Hypergraph partition.

be assigned to non-local datacenters, thereby resulting in cross-domain data transfer. While considering more than one replica for each file, the task assignment is made more complicated by the introduction of data replica selection.

We combine the replica selection and task placement problems using an *augmented task-file-DC hypergraph*. In particular, first, we extend the task graph to create an *augmented task graph* that leverages the dependency relations between tasks and input files and also the dependency relations between file replicas and datacenter locations. Next, we model the augmented task graph as a hypergraph, which allows us to solve the data replica selection problem and task placement problem simultaneously, and distributes the computational loads among datacenters to reduce the amount of data transfers.

#### 4.1.1 Augmented Task Graph Construction

Since each $f \in N^F$ may have multiple replicas, which are distributed over more than one datacenter, the dependency relationship between $f$ and datacenters is used to extend the task graph. That is, the augmented task graph $G^{AG} = (N^{AG}, E^{AG})$ is constructed as below,

$$N^{AG} = N^T \cup N^F \cup N^S;$$

$$E^{AG} = E^V \cup \{(f, n) | f \in N^F, n \in N_f^S\}$$

where $N^{AG}$ consists of all nodes belonging to $N^T$, $N^F$ and $N^S$, and $E^{AG}$ is obtained by connecting $f \in N^F$ to the substrate nodes in $N_f^S$ using meta edges, where $N_f^S$ refers to the set of datacenters storing replicas of $f$. Fig. 4 illustrate the augmented task graph for the example shown in Fig. 2.

#### 4.1.2 Hypergraph Partitioning

We use a hypergraph to model the augmented task graph. A hypergraph $H = (V, E)$ is defined as a set of vertexes $V$ and a set of nets (hyperedges) $E$. Every net $n_j \in E$ consists of a subset of vertexes. Weights can be associated with the vertexes and costs can be associated with the nets. The *k-way hypergraph partitioning problem* is defined as [19]: given a hypergraph $H = (V, E)$, find a *k-way partition* $V \rightarrow P$ that maps the vertexes of $H$ to one of $k$ disjoint partitions such that some cost function is minimized and the given partitioning constraints are satisfied. Let $P = \{H_1, H_2, ..., H_k\}$ be a *k-way partition* of $H$, then $\forall i, j \in [1..k]$, we have $H_i \cap H_j = \emptyset$ and $H_1 \cup H_2 ... \cup H_k = H$. We call a hyperedge $e$ of hypergraph is a *cut* if, with respect to a particular partition, its vertexes are mapped to more than one partition.

We next show that, the solution to the *k-way hypergraph partitioning problem* that minimizes the cut size (defined over the cut nets), while meeting a given balance criterion, can be applied to data replica selection and task placement. In summary, we design four techniques to do this:

**(1) Introducing unit-weighted "virtual tasks"**: To remove the balance criterion in the original hypergraph partitioning, we introduce a number of "Virtual Tasks" (denoted by $N^*$) with each of them requiring a single core, for ease of hypergraph partitioning. Tasks belonging to $N^*$ are only used in the scheduling process and not in the real execution. As a result, all "Virtual Tasks" are independent: each of them takes no input and generates no output, and it takes no time to execute. If the required number of cores for all real tasks (denoted by $R$) are less than the supply of all datacenters (denoted by $S$), the number of "Virtual Tasks" would be $S - R$.

In our hypergraph $H = \{V, E\}$, each vertex in $V$ is also a node in $N^T \cup N^S \cup N^*$, i.e., $V = N^T \cup N^S \cup N^*$. In other words, there exist three types of vertexes in $V$: the real task vertexes, corresponding to tasks in $N^T$; the virtual task vertexes, corresponding to "Virtual Tasks" in $N^*$; and the datacenter vertexes, corresponding to datacenters in $N^S$. Hence, $|V| = |N^T| + |N^S| + |N^*|$.

**(2) Negative weighted hypergraph nodes representing datacenters**: For a vertex $v_t \in N^T \cup N^*$, we set its weight to the capacity requirement of the task, i.e., $w(v_t) = h_{v_t}$. For a vertex $v_s \in N^S$, we set its weight to the negative of the capacity of datacenter, i.e., $w(v_s) = -R_{v_s}$. A net $n_j \in E$ represents the file $f_j$, it contains the task vertexes in $N_{f_j}^T$ and the datacenter vertexes in $N_{f_j}^S$. We set the cost of $n_j$ to the size of the file, i.e., $c(n_j) = s_{f_j}$. Note that tasks in $N^*$ have no input, so there exist no nets connecting tasks in $N^*$ and any files.

**(3) Double-counting free cut-cost function**: Each partition includes a group of task nodes and a single datacenter node. It suggests that the tasks in the partition should be scheduled in the datacenter, leading to the minimum data transfer across partitions, hence we let $k = |N^S|$ in our hypergraph partitioning settings. Let $\Lambda_j$ be the connectivity set of a net $n_j$ (representing file $f_j$), which is the set of partitions that $n_j$ connects, and $\lambda_j = |\Lambda_j|$ is the number of partitions it connects. Then, $f_j$ needs to be transferred for $\lambda_j - |N_{f_j}^S|$ times, where $|N_{f_j}^S|$ denotes the number of datacenters storing $f_j$. Therefore, we define the cost function *Cut(P)* as below:

$$Cut(P) = \sum_{n_j \in P} c(n_j)(\lambda_j - |N_{f_j}^S|) \tag{19}$$

**(4) Zero-approaching balance constraint**: The objective of hypergraph partitioning seeks to minimize $Cut(P)$,

while maintaining a balance on the part weights, i.e.,

$$\sum_{v \in H_j} w(v) \leq \epsilon, \qquad \forall H_j \in P \tag{20}$$

where $\epsilon$ is a predetermined small constant close to 0, implying the imbalance ratio. Clearly, we have $\epsilon < min\{h_{v_t}\}$.

We now show that, with the four techniques designed above, each partition contains one and only one datacenter vertex. Therefore, with the above constraints, the capacity requirements of tasks assigned to the same datacenter will not exceed the datacenter's capacity.

**Theorem 2.** *Given an augmented task hypergraph $H = \{V, E\}$: $V = N^{AG} \cup N^*$, $E = E^{AG}$, suppose $\forall v_t \in N^T$, $w(v_t) = c_{v_t}$, $\forall v_f \in N^F$, $w(v_f) = 0$ and $\forall v_s \in N^S$, $w(v_s) = -R_{v_s}$, then for a $|N^S|$-way partition $P = \{H_1, H_2, ..., H_{|N^S|}\}$, balance constraint: $\forall H_j \in P$, $\sum_{v \in H_j} w(v) \leq \epsilon$, ensures that each partition contains one and only one datacenter vertex.*

*Proof:* In the $|N^S|$-way partitionment, to ensure $\forall H_j \in P$, $\sum_{v \in H_j} w(v) \leq \epsilon$, each partition must have at least one vertex whose weight is negative. Because $w(v_s) < 0$, $w(v_t) > 0$ and $w(v_f) = 0$, each partition must have at least one datacenter vertex. As all vertexes are divided into $|N^S|$ partitions, and there are just $|N^S|$ vertices associated with negative weight, each partition contains one and only one datacenter vertex. Proof end. □

**Corollary 1.** *The cut cost function $Cut(P) = \sum_{n_j \in P} c(n_j)(\lambda_j - |N^S_{f_j}|)$ directly derives the amount of data transferred among datacenters.*

*Proof:* This can be derived using Theorem 2: since each partition contains one and only one datacenter vertex, there must be $\lambda_j - |N^S_{f_j}|$ partitions who does not have $f_i$ locally. Hence, the WAN data transfer volume is $\sum_{n_j \in P} c(n_j)(\lambda_j - |N^S_{f_j}|)$. □

**Theorem 3.** *Given an augmented task hypergraph $H = \{V, E\}$: $V = N^{AG} \cup N^*$, $E = E^{AG}$, the optimal partition $P = \{H_1, H_2, ..., H_{|N^S|}\}$ to the $|N^S|$-way hypergraph partitioning problem that minimizes $Cut(P)$ while satisfying the balance constraint: $\forall H_j \in P$, $\sum_{v \in H_j} w(v) \leq \epsilon$, also minimizes the volume of WAN data transfer.*

*Proof:* We prove this by contradiction. Assume that task $i$ is placed into $H_a \in P$, but it should be placed into $H_b$ if we solely minimize the WAN data transfer. Since the vertexes belonging to $N^*$ are isolated points without connection to any other vertexes, they can be simply placed in an arbitrary partition, as long as their aggregated capacity requirements do not exceed the datacenter's capacity (i.e., following Formula 20). Therefore, by placing $i$ into $H_b$ and moving $R_i$ number of "virtual tasks" from $H_b$ to $H_a$, we get a new $|N^S|$-*way partitionment* $P'$, where $Cut(P') < Cut(P)$ and the balance constraint $\forall H_j \in P$, $\sum_{v \in H_j} w(v) \leq \epsilon$ is still satisfied. This contradicts the fact that $P$ is the optimal partition. Thus, the $|N^S|$-way partition of $H$ also minimizes the volume of WAN data transfer. □

The hypergraph partitioning problem is known to be NP-hard [20], and it can be solved using the Multilevel Fiduccia-Mattheyses (MLFM) framework [21], that has three phases (i) Coarsening: coarsen the initial hypergraph into a sequence of smaller hypergraphs; (ii) Top-level partitioning: partition the smallest hypergraph; (iii) Refinement: project a coarse partitioning to a finer hypergraph and improve the partitioning using a refinement method. Since the coarsening phase takes a linear time complexity: $O(|V|)$ [22] and the Fiduccia-Mattheyses heuristic in top-level partitioning phase takes time $O(|E|)$ [23], the overall time complexity of MLFM is $O(|V|+|E|+\mu|E|)$ [19], where $\mu$ is the number of levels used in refinement phase, and usually only a small number of levels are needed in practice.

We implement our approach based on *customized Sandia's Zoltan toolkit* [24]. However, the latest Zoltan v.3.83 only supports a cost function (i.e., $Cut(P)$) defined in function *Zoltan_PHG_Compute_ConCut*(), which calculates $Cut(P)$ of a partition in the following way:

$$Cut(P) = \sum_{n_j \in P} c(n_j)(\lambda_j - 1)$$

Compared with (19), this does not meet our needs because we use data replicas to reduce data transfer on WAN. Hence, we need a customized partitioning supporting our cost function (19) and balance criterion (20). We improve the Zoltan toolkit by implementing mechanisms distinguishing the task vertexes and datacenter vertexes, and a new *Zoltan_PHG_Compute_ConCut_MultiReplicas*() function is implemented to compute $Cut(P)$ using (19). We also define a new function of *Zoltan_Set_HG_CS_Custom_Fn*() to pass the parameter of the number of replicas (i.e., $fileCopy$) to Zoltan, as shown in the listing below.

```
int Zoltan_Set_HG_CS_Fn(ZZ *zz,
ZOLTAN_HG_CS_FN *fn, void *data, int *fileCopy)
{
  zz->Get_HG_CS = fn;
  zz->Get_HG_CS_Data = data;
  zz->fileCopy = fileCopy;
  return ZOLTAN_OK;
}
```

Fig. 4 also describes the partitioning results for the example in Fig. 2. That is, task $T_2$ and $T_3$ are assigned to $S_1$, $T_1$ is assigned to $S_3$, $T_4$ and $T_5$ are assigned to $S_4$, generating two WAN data transfers: sending $f_1$ from $S_1$ to $S_3$, and sending $f_5$ from $S_1$ to $S_4$. The total WAN data transfer volume is 2000, which is far less than the other algorithms. The makespan is 104.3, which is also much earlier than the others.

### 4.2 Flow Routing and Bandwidth Allocation

Given the task scheduling results, we select the replica that requires the least number of hops to reach as the input for each task. Then, we orchestrate the generated data transfers for minimizing the makespan, i.e., the completion time of

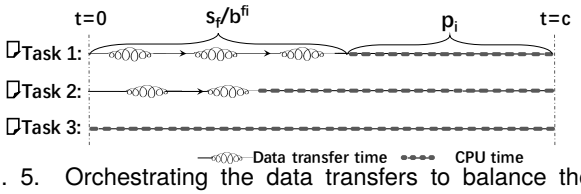



Fig. 5. Orchestrating the data transfers to balance their completion time.

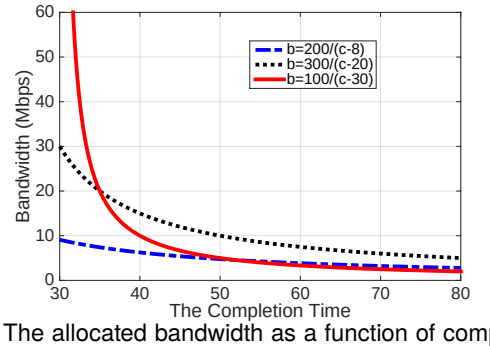



Fig. 6. The allocated bandwidth as a function of completion time

the slowest task. Since it is difficult to have full control over network bandwidth resources in the traditional network architecture, we assume a SDN-enabled geo-distributed cloud environment [9], which enables customization of network operations by decoupling the network control plane and data plane, supporting accurate bandwidth allocation to flows. While existing approaches usually address the task placement [13], [14] and flow routing [9], [25] separately, thus achieving sub-optimal performance, we present a coordination mechanism between tasks and WAN traffic, which allocates network resources closely following the guidelines of task requirements.

Generally, tasks with longer CPU time are supposed to obtain more bandwidth than the others, unless their input data sizes are small. To complete a task $i$ before $c$, we can compute the required bandwidth for transferring flow $fi$ as below,

$$\frac{s_f}{b^{fi}} + p_i \leq c \Rightarrow b^{fi} \geq \frac{s_f}{c - p_i}, \quad \forall fi \in E^V \tag{21}$$

The makespan is at least equal to the maximum CPU time, i.e., $c \geq \max_{\forall i \in N^T}(p_i)$, where the equation holds only when the slowest task accesses its input data locally. Let $p_{max} = \max_{\forall i \in N^T}(p_i)$. If $c = p_{max}$ and $p_{max} > p_i$, then the deadline $c$ is not violated by $i$ only when $b^{fi} \geq s_f/(p_{max} - p_i)$ is established. Fig. 5 illustrates three examples of tasks: task 3 has the longest CPU time; task 1 (or task 2) must request bandwidth at least $s_f/(p_3 - p_1)$ (or $s_f/(p_3 - p_2)$) to ensure the makespan $p_3$. However, since the network may not be able to provide sufficient bandwidth resources, the makespan may be increased from $c$ to $c_1$ if the bandwidth that is allocated to flow $fi$ is decreased by $\Delta b^{fi}$:

$$\Delta b^{fi} = \frac{s_f}{c - p_i} - \frac{s_f}{c_1 - p_i} \tag{22}$$

The flow routing problem can be modeled as a Multi-commodity Flow Problem (MCF) that provides a multi-path routing solution for each flow using optimal linear programming algorithms [26]. Our goal is to find a solution with a maximum allocation of bandwidth for each commodity with respect to their utility functions (Formula 21). The MCF problem can be solved directly, but the linear programming (LP) optimal solution is expensive and does not scale well. Hence, we borrow the basic design principles of an efficient traffic engineering algorithm from B4 [9], and adapt it to our settings as follows: taking the makespan as the fair share level and allocating bandwidth following the utility function of 21. The key idea is that, the bandwidth allocation step is repeated by progressively decreasing a pre-selected large $c$. Everytime we decrease $c$, the newly requested bandwidth by each flow is computed using Formula 22, and is allocated from the current shortest routing path. Fig. 6 illustrates three examples of the functional relationship between the requested bandwidth and the makespan. The requested bandwidth by each flow increases at different slopes as the makespan decreases.

To improve the network utilization, although the makespan cannot be further reduced after a flow $fi$ gets bandwidth more than $b^{fi} = s_f/(p_{max} - p_i)$, we do not stop the bandwidth allocation process but continuously allocate more until the link is fully utilized.

Algorithm 1 presents our design of routing and bandwidth allocation (RBA) algorithm. First, we initialize the makespan $c$ with a large value $MAX$ (Line 6). Then, we compute the required bandwidth $b^{ij}$ for each flow using Formula 21, and allocate them with $b^{ij}$ from their respective shortest path for satisfying $c$ (Line 7-11). The bandwidth allocation step is repeated by progressively decreasing $c$ (Note that the value of $c$ depends on the real problem settings. In our experiments, we initialize $c = 20,000$ and the step size of decrement is set to 500.): everytime we decrease $c$, the allocated bandwidth to each flow increases by $\Delta b$, which is derived by Formula 22 (Line 13-14). The increased $\Delta b$ to each flow causes updates to the current path's used bandwidth and each network link's remaining bandwidth (Line 15-18). If the current path of flow $ij$ cannot provide more bandwidth for satisfying $\Delta b$, we choose to allocate more bandwidth from the next shortest path in $P_{ij}[N]$ (Line 20-22). Likewise, if the available bandwidth for a flow runs out, it is removed from the allocation process (Line 23-24). The time complexity of RBA is $O(\phi|E^V|)$, where $\phi$ denotes the number of times we decrease $c$ and $|E^V|$ is the maximum number of flows generated by tasks.

**Theorem 4.** *The time complexity of HPS+ is* $O(|N^T|+|N^S|+|N^*| + \mu|N^F| + \phi|E^V|)$.

*Proof:* Since the time complexity of hypergraph partitioning is $O(|V| + |E| + \mu|E|)$ and the time complexity of RBA is $O(\phi|E^V|)$, and we have $|V| = |N^T| + |N^S| + |N^*|$ and $|E| = |N^F|$, the overall time complexity is $O(|N^T| + |N^S| + |N^*| + \mu|N^F| + \phi|E^V|)$.

□

**Algorithm 1:** Routing and Bandwidth Allocation Algorithm (RBA)

**Input :** $N^V = N^T \cup N^F$, $E^V$, $G^S = (N^S, E^S)$;
**Output** Mapping of flows in $E^V$ to $E^S$;
**:**

**[Parameters]:**
**Time** *c;* {*the makespan.*}
**Flow** $ij$; {*data flow from data i to task j.*}
**Path** $P_{ij}[N]$; {*The N shortest paths for flow ij.*}

**[Routing Phase]:**
Initialize $c \leftarrow MAX$;
**for** *each data flow* $ij$ **do**
  $ij.curP = P_{ij}[0]$; // from the 1st path
  Compute for $b^{ij}$ using Formula 21;
  Add $b^{ij}$ to *ij.curP.usedbw;*
  Subtract $b^{ij}$ from $E^S.link.remainingbw$;
**do**
  Decrease $c$;
  Compute $\Delta b^{ij}$ for each flow using Equ. 22;
  **for** *each unfinished flow* $ij$ **do**
    Add $\Delta b^{ij}$ to *ij.curP.usedbw;*
    Subtract $\Delta b^{ij}$ from $E^S.link.remainingbw$;
  update all *ij.curP.remainingbw;*
  **if** *ij.curP cannot provide more bw satisfying* $ij.\Delta$ **then**
    $ij.usedPathSet \leftarrow ij.curP$;
    *ij.curP*++; // next path
  **if** *ij has no more avaliable bw* **then**
    *move ij to the finished flows set.*
**while** *No more bw left or all bw demands are met*;
Quantize the allocated bw to each flow.

TABLE 2
Configurations of datacenters in CVO.

| Configuration | Beijing | Nanjing | Shanghai | Yunnan | Xinjiang |
|---|---|---|---|---|---|
| Data (TB) | 55 | 50 | 30 | 10 | 5 |
| Cores (#) | 900 | 690 | 850 | 350 | 160 |
| Core rating (MIPS) | 1330 | 1166 | 1200 | 1140 | 1000 |

## 5 Performance Evaluation

### 5.1 Experimental Setup

**Cloud system:** We simulate two real networks: the China-VO network and the Google B4's network. In the China-VO network, the cloud system has 5 datacenters and 7 WAN links [8] We configure the network based on the real parameters from China-VO, including datacenters' configurations as shown in Tab. 2 and the WAN bandwidth capacity of all links are set by $1GE$. In the Google network, the cloud system has 12 datacenters and 19 WAN links [9]. We do not have the capacity information for it, but simply simulate the system with random computing capacity from a uniform distribution with range $[2, 32] \times 10^3$, and set the bandwidth capacity the same as China-VO network.

TABLE 3
Performance comparison (B4 network).

| Metric | | File Transfer Volume | | | | Makespan | | | |
|---|---|---|---|---|---|---|---|---|---|
| $1 - \frac{hps+}{alg.}$ | | GS | HPS | CDS | Flutter | GS | HPS | CDS | Flutter |
| *ccr=0.1* | max | 0.23 | 0.33 | 0.31 | 0.24 | 0.19 | 0.21 | 0.30 | 0.21 |
| | mean | 0.17 | 0.27 | 0.22 | 0.19 | 0.11 | 0.15 | 0.25 | 0.10 |
| | min | 0.09 | 0.22 | 0.08 | 0.12 | 0.01 | 0.11 | 0.17 | 0.01 |
| *ccr=1* | max | 0.20 | 0.29 | 0.28 | 0.22 | 0.09 | 0.13 | 0.24 | 0.09 |
| | mean | 0.16 | 0.26 | 0.20 | 0.18 | 0.04 | 0.10 | 0.18 | 0.03 |
| | min | 0.09 | 0.23 | 0.05 | 0.13 | 0.00 | 0.07 | 0.13 | -0.01 |
| *ccr=10* | max | 0.24 | 0.34 | 0.30 | 0.26 | 0.09 | 0.12 | 0.25 | 0.12 |
| | mean | 0.13 | 0.25 | 0.20 | 0.16 | 0.03 | 0.09 | 0.18 | 0.05 |
| | min | 0.04 | 0.20 | 0.01 | 0.06 | -0.02 | 0.04 | 0.08 | -0.02 |

TABLE 4
Performance comparison (CVO network).

| Metric | | File Transfer Volume | | | | Makespan | | | |
|---|---|---|---|---|---|---|---|---|---|
| $1 - \frac{hps+}{alg.}$ | | GS | HPS | CDS | Flutter | GS | HPS | CDS | Flutter |
| *ccr=0.1* | max | 0.42 | 0.53 | 0.39 | 0.40 | 0.29 | 0.36 | 0.26 | 0.27 |
| | mean | 0.29 | 0.32 | 0.25 | 0.28 | 0.24 | 0.26 | 0.20 | 0.21 |
| | min | 0.11 | 0.05 | 0.09 | 0.08 | 0.10 | 0.04 | 0.07 | 0.05 |
| *ccr=1* | max | 0.39 | 0.52 | 0.34 | 0.40 | 0.33 | 0.39 | 0.29 | 0.29 |
| | mean | 0.30 | 0.32 | 0.27 | 0.29 | 0.24 | 0.26 | 0.21 | 0.21 |
| | min | 0.12 | 0.02 | 0.09 | 0.08 | 0.11 | 0.02 | 0.09 | 0.07 |
| *ccr=10* | max | 0.36 | 0.49 | 0.32 | 0.35 | 0.31 | 0.36 | 0.30 | 0.28 |
| | mean | 0.29 | 0.31 | 0.25 | 0.27 | 0.23 | 0.25 | 0.20 | 0.19 |
| | min | 0.12 | 0.04 | 0.10 | 0.10 | 0.11 | 0.03 | 0.08 | 0.06 |

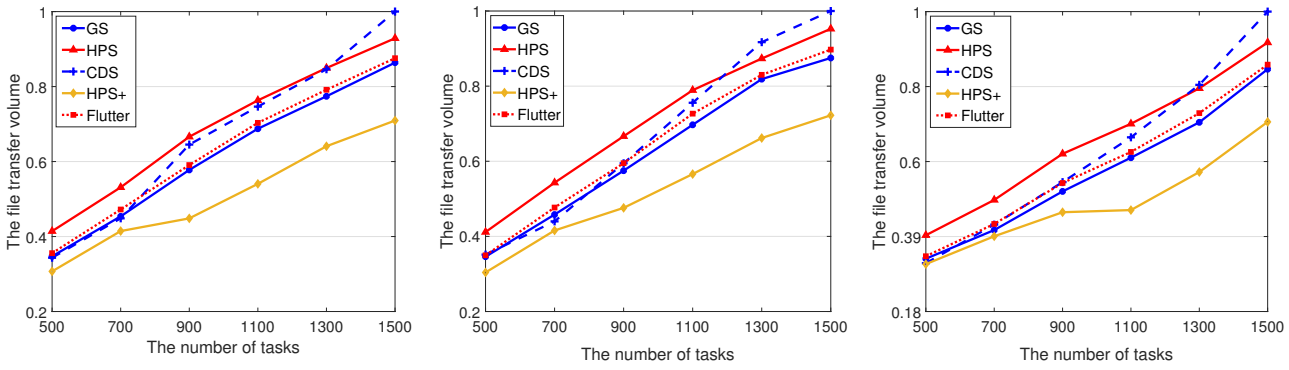
**Workloads:** We simulate at most 2000 tasks and schedule them to datacenters. Each task requests a certain amount of computation capacity, which is randomly generated from range $[1, 10]$. The CPU time of a task $j$ (i.e., $p_j$) is selected randomly from range $[10, 1000]$ seconds, following Facebook's experiences that more than 90% of their tasks are completed within 1000 seconds [27]. We also evaluate the performance of our algorithms under different communication and computation loads (denote by *CCR* the communication to computation ratio), and the input files of each task is determined by $CCR \times p_j$. We create 2000 files in different sizes and distribute them to China-VO datacenters, and distribute 8000 files to the Google datacenters. The size of each file is selected randomly from a uniform distribution with range $[0.2GB, 5GB]$ by studying the history data from astronomy.

**Performance comparison:** We compare our proposal with four scheduling algorithms: Greedy, CDS [13], HPS [8], Flutter [14]. Given the output schedule by these algorithms, they default to utilizing the traditional TCP/IP-based network protocol for data transfer, but do not take into account the coordination between task scheduling and flow transfer. To be fair, we also apply our RBA algorithm into their bandwidth allocation. All results are normalized to 1 by dividing their values by the maximum in its group of experiments.

### 5.2 Experimental Results

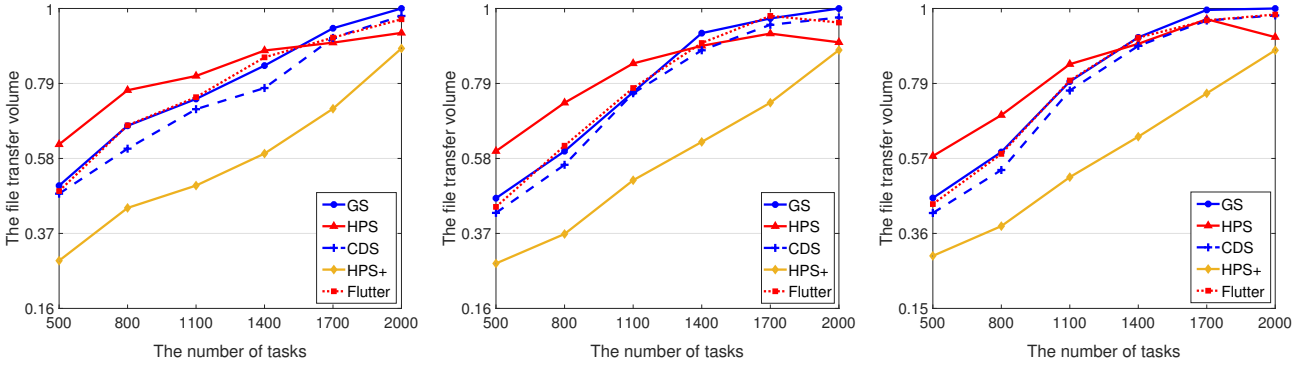
#### 5.2.1 The Data Transfer Volume

Firstly, we compare HPS+ to GS, CDS, HPS and Flutter on the metric of data transfer volume. Fig. 7 and Fig. 8 highlight the total file transfer volume as a function of increasing number of tasks, under different *CCR* settings. We can see that the total file transfer volume generated




(a) $ccr = 0.1$ (b) $ccr = 1$ (c) $ccr = 10$

Fig. 7. The total data transfer volume generated by algorithms in the B4 network (Normalized to $\leq 1$).



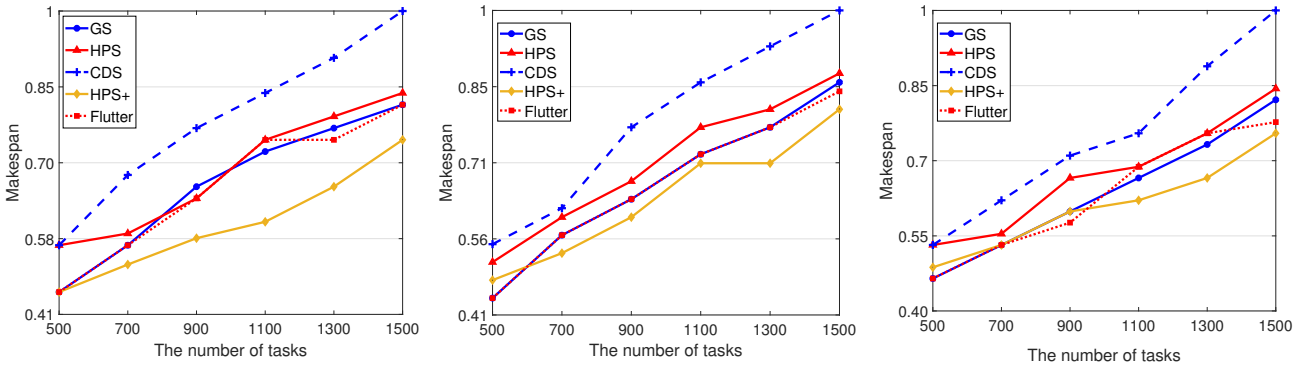

(a) $ccr = 0.1$ (b) $ccr = 1$ (c) $ccr = 10$

Fig. 8. The total data transfer volume generated by algorithms in the CVO network (Normalized to $\leq 1$).

by the algorithms increases with the number of tasks. In all the cases, HPS+ performs significantly better than the other algorithms. Tab. 3 shows that, on B4 network, HPS+ can reduce the file transfer volume by upto 33%, when $CCR = 0.1$. Likewise, in case of CVO network (Tab. 4), HPS+ can reduce the file transfer volume by upto 53%. Even for the data-intensive tasks (i.e., $CCR = 1, 10$), HPS+ still performs a substantial reduction in WAN data transfer volume. In particular, with $CCR = 10$ in CVO network, HPS+ outperforms GS, HPS and CDS on the file transfer volume metric by an average of 29%, 31%, 25%, and 27%, respectively. Among the algorithms, HPS does not perform well, because its hypergraph partition process not only tries to minimize the data transfer volume, but also optimizes the load balance between groups. Therefore, a load-balanced task group could increase the file transfer volume. CDS algorithm outperforms HPS, but the gap gradually narrows as the number of tasks increases. When the task load exceeds 1100 in B4 case (or 1500 in CVO case), CDS algorithm generates more data transfer due to inaccurate task scheduling order. Flutter provides comparable performance to GS on data transfer volume, since Flutter's exclusive network bandwidth assumption limits its performance in shared networks.

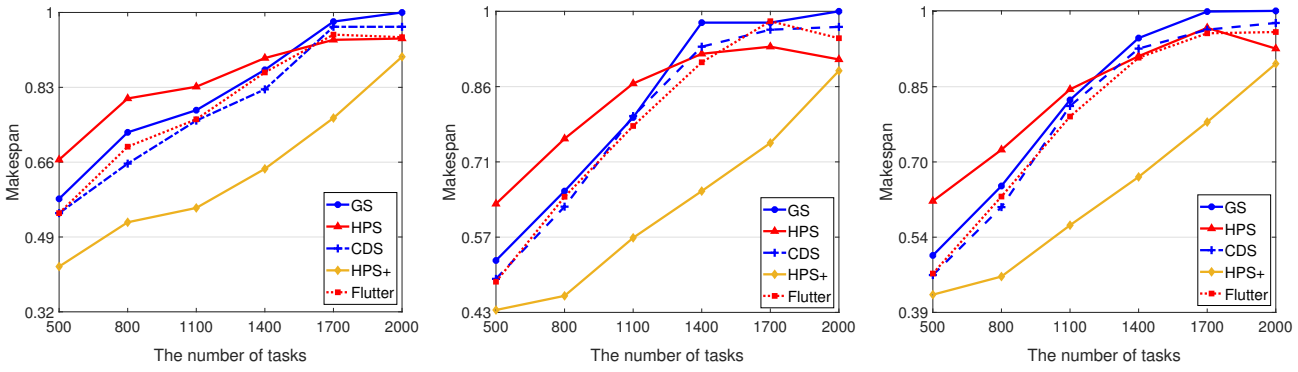
### 5.2.2 *The Makespan*

The reduction in the amount of WAN traffic can reduce the makespan. Fig. 9 and Fig. 10 show the makespan as a function of increasing number of tasks on B4 network and CVO network, respectively. We find that, on B4 network, HPS+ outperforms GS, CDS, HPS and Flutter on the makespan metric by an average of 11%, 15%, 25% and 10%, respectively, when $CCR = 0.1$. While on CVO network, HPS+ reduces the makespan by an average of 24%, 26%, 20%, 21%, respectively. Tab. 3 and Tab. 4 also show that HPS+ brings a substantial reduction in makespan for more data-intensive task load (i.e., $CCR = 1, 10$). In particular, when $CCR = 10$, HPS+ can reduce the makespan by upto 30% in B4 network, and by upto 39% in CVO network. We also see that while CDS generates shorter makespan than GS and HPS in the CVO case, it performs much worse than them in the B4 case. The reason is that CDS requires a much longer warming times to maximize its modularity in the larger scale network of B4. The gap between HPS+ and HPS gradually narrows as the increase of the number of tasks, especially in CVO network, due to the natural support for high computation requirements by HPS. We do not compare their performance under settings with more than 2000 tasks, because the overall computation requirements generated using random numbers are exceeding the overall supply by datacenters, which is not supported by HPS+.



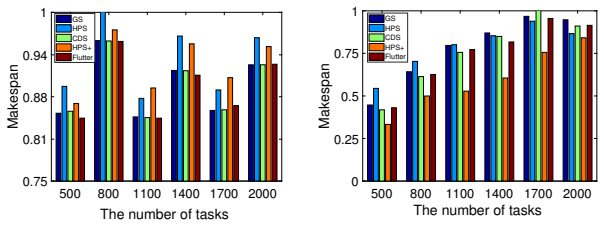

(a) $ccr = 0.1$ (b) $ccr = 1$ (c) $ccr = 10$

Fig. 9. The makespan generated by algorithms in the B4 network (Normalized to $\leq 1$).



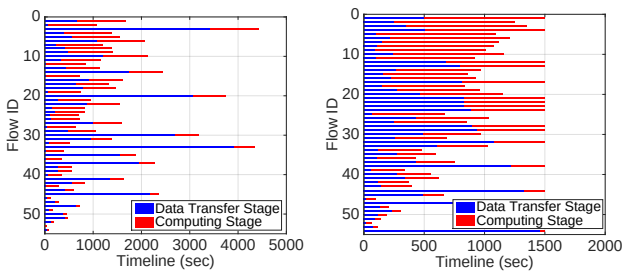

(a) $ccr = 0.1$ (b) $ccr = 1$ (c) $ccr = 10$

Fig. 10. The makespan generated by algorithms in the CVO network (Normalized to $\leq 1$).




(a) In exclusive network (b) In shared network

Fig. 11. The makespan when cpu requirments of all tasks are set to 1 and *ccr=1* in CVO network (Normalized to $\leq 1$).




(a) (b)

Fig. 12. The timeline of file transfer stage and computing stage: (a) in traditional TCP/IP network; (b) using RBA algorithm in SDN.

Since the system settings by Flutter is slightly different with ours, we also did another group of experiments in CVO network by setting $h_i = 1$, $\forall i$, and assuming each data flow is transferred by an exclusive routing path. Fig. 11 shows that Flutter indeed outperforms other algorithms in the exclusive network, but HPS+ generates the shortest completion time in the more real shared network.

We also illustrate the timeline (including data transfer stage and computing stage) of 55 data flows generated by an example run of the scheduler in Fig. 12. We see that the makespan in traditional TCP/IP network performs much larger variance than that in SDN network. Although the computing stage of each task takes the same time in both network scenarios, their data tranfer time varies significantly. In traditional TCP/IP network, bandwidth is allocated in a "best-effort" way, hence the task with long computing stage may not be able to recieve more bandwidth to reduce its file transfer stage (e.g., Flow ID #3). In SDN, however, the tasks with longer computing stage are always prioritized for more bandwidth, due to the coordination between task placement and flow routing, and the exact bandwidth allocation by RBA algorithm. Therefore, the makespan becomes much shorter.

### 5.2.3 *Algorithm Running Time*

Fig. 13 shows the running time of the algorithms. We see that Flutter takes far more time than the others, and even reaches 6,089 seconds when the number of tasks is

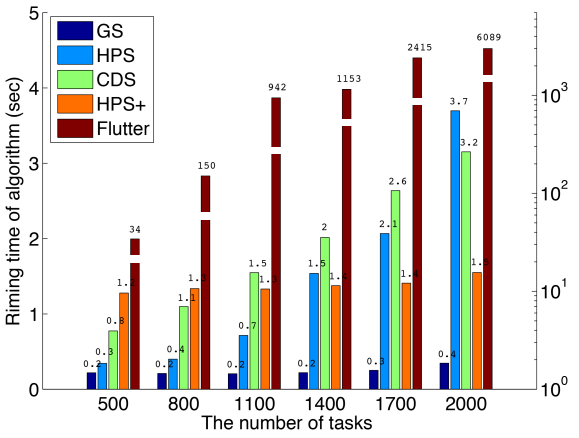


Fig. 13. The running time of algorithms

2,000, while the others only takes less than 4 seconds. This is primarily due to the low-efficient ILP solving process. The time complexity of GS, CDS, HPS and HPS+ are $O(|N^V||N^S|)$, $O(|N^V||N^S|)$, $O(|E^V||N^S||N^V|^2 + O(|E^V||N^S|^2|N^V|)$ and $O(|N^T| + |N^S| + |N^*| + \mu|N^F| + \phi|E^V|)$, resepctively. Although GS and CDS have the same time complexity, CDS takes more time than GS due to its complex computation on Q-measure. HPS takes a lot of time in its task reassigning phase, and its running time increases significantly as we increase the number of tasks. HPS+'s running time does not vary much over the number of tasks, because it does not have the reassigning phase, and the total number of tasks in the hypergraph actually does not change due to the created "virtual tasks" ($N^*$).

## 6 Related Work

### 6.1 Geo-distributed data analytics

Big data analytics across multiple data centers are receiving increasing attention in recent days [28]. One major challenge for geo-distributed data analytics is the high costs and latency of links spanning multiple domains, which could be an order of magnitude higher than the intra-domain links. Li et al. [29] jointly consider input data movement and task placement, to minimize the WAN traffic generated by Mapreduce jobs. However, their proposal cannot solve our problem because they did not study the data sharing and network sharing issue in scheduling. Heintz et al. [30] only study the *grouped aggregation* primitive in geo-distributed data analytics, and jointly minimize both staleness and WAN traffic by utilizing both edge and central resources in a carefully coordinated manner. Awan [31] is a two-level architecture resource manager for edge cloud environment, and enables each computing framework's jobs to be scheduled with high locality to reduce WAN traffic. Iridium [32] achieves low query latency by optimizing placement of both data and tasks of the queries, while our model assumes datasets are previously distributed over datacenters. Hung et al. [33] propose a job scheduling algorithm called SWAG to reduce the makespan through coordinating job scheduling across datacenters with low overhead. Their proposal completely avoids WAN data transfer by dividing jobs into arbitrary pieces, each of which is assigned to the datacenter with its input dataset. Clearly, WAN data transfer could reduce the makespan if the workloads vary significantly across datacenters.

In case of scheduling data-intensive flows in multi-DC environment, Zhang et al. [34] optimize the initial data distribution, task placement and task replication in a combined way to reduce the data transfer volume. For service placement problem, Steiner et al. [35] and Selimi et al. [36] also propose bandwidth-aware algorithms to maximise the WAN bandwidth gain. In contrast with them, we not only reduce the WAN traffic, but also present coordination mechanism to minimize the makespan. Xu and Li [37], Jiang et al. [38] and Narayana et al. [39] also take into account the route selection in coordination with task placement, but they do not support precise bandwidth allocation.

### 6.2 Task scheduling

Many algorithms [40]–[44] have been presented to schedule a large collection of file-sharing tasks onto heterogeneous clusters with a goal of minimizing the total completion time by considering the system's heterogeneity such as networks and CPUs. Giersch et al. [40], [42] design a collection of heuristics to reduce the total execution time with lower computational costs. In [41], they further deal with a more general scheduling problem in a platform similar with ours. This work establishes the NP-completeness of the scheduling problem and design several new heuristics, including an extension of the min-min heuristic. Based on the work above, Kaya and Aykanat [43], [44] utilize hypergraph to model the tasks, files and their interactions and thereby formulate the total communication and computation cost. They firstly find a lower and upper bounds on the turnaround time, then iteratively refine the task assignments by closing the gap between lower and upper bounds. In [43], they assume a master-slave paradigm in the computing model, and the master/server is as a repository for all files. In [44], they turn to consider distributed repositories, and all repositories are not necessary close to processors. However, in our model, all files have been distributed inside datacenters, i.e., we neither adopt a centralized repository for all files, nor let files be dispersed far from processors. Therefore, their proposal does not apply to our system.

Çatalyürek et al. [45] model the workflow as a hypergraph and with a hypergraph-partitioning-based formulation, they propose a heuristic to reduce the file transfer in the execution of workflows. Khanna et al. [46] also utilize the hypergraph partitioning to minimize total volume of file transfers and maintaining a balance on the loads of the processors in a grid system. They deal with an application consisting of file-sharing otherwise independent tasks running on a set of homogeneous processors connected to a set of storage nodes through a uniform (homogeneous) network. Then, they used a hypergraph to model the application which was partitioned into groups – one group to one processor – using hypergraph partitioning. However, if

the capability of computing nodes is not homogeneous, this method would be ineffective. Venugopal et al. [47] apply a Set Covering Problem (SCP)-based matching heuristic to match independent tasks to resources in a way that they explore all sets of job-resource combinations and select the best one with the Minimum Completion Time (MCT) of each job, thereby minimizing the makespan. However, they do not take the task-task relations into consideration which may result in shared data sets being retrieved multiple times.

## 7 Conclusions and Future Work

In this paper, we address the problem of optimizing geo-distributed data analytics with coordinated task scheduling and flow routing, and present the design of HPS+. In HPS+, we utilize hypergraph to model the combined task-data dependencies and data-datacenter dependencies, and seek for a possible schedule by solving the hypergraph partitioning problem. We design four techniques to ensure that the optimal *k-way partitionment* of the hypergraph can generate a schedule with minimum WAN data transfer. Given the generated task schedule, we also propose a traffic engineering algorithm, which is able to allocate network resources for all data flows in a coordinated manner with the schedule result. In this way, tasks with longer computation stage will be likely to get more bandwidth. Experiments with two cloud systems show that HPS+ can significantly reduce the WAN traffic, and thereby reduce the makespan.

The current implementation of HPS+ does not take into account the de-allocation of network bandwidth. In the future, we would like to further improve the network resource usage and reduce the makespan through allocating and de-allocating network resources. We will also continuously deploy and optimize its implementation in the astronomical cloud platform to improve its efficiency.

## References


[1] G. Andronico, V. Ardizzone, R. Barbera, B. Becker, R. Bruno, A. Calanducci, D. Carvalho, L. Ciuffo, M. Fargetta, and E. Giorgio. e-infrastructures for e-science: a global view. *Journal of Grid Computing*, 9:155–184, 2011.

[2] D.A. Reed. Grids, the TeraGrid and beyond. *Computer*, 36:62–68, 2003.

[3] F. Ping, X. Li, C. McConnell, and R. Vabbalareddy. Towards optimal data replication across data centers. In *31st Inter. Conf. on Distributed Computing Systems Workshops*, pages 66–71. IEEE, 2011.

[4] D. Yuan, Y. Yang, X. Liu, and J. Chen. A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems*, 26:1200–1214, 2010.

[5] R.L. Grossman, Y. Gu, M. Sabala, and W. Zhang. Compute and storage clouds using wide area high performance networks. *Future Generation Computer Systems*, 25:179–183, 2009.

[6] J. Parikh. keynote speech: Data infrastructure at web scale. In *VLDB'13*, 2013.

[7] J. Becla, K.T. Lim, S. Monkewitz, M. Nieto-Santisteban, and A. Thakar. Organizing the extremely large LSST database for real-time astronomical processing. In *17th Annual Astronomical Data Analysis Software and Systems Conference (ADASS 2007), London, England*, pages 23–26, 2007.

[8] L. Yin, J. Sun, L. Zhao, C. Cui, J. Xiao, and C. Yu. Joint scheduling of data and computation in geo-distributed cloud systems. In *CCGrid*, pages 657–666. IEEE, 2015.

[9] Sushant Jain, Alok Kumar, Subhasree Mandal, and et al. B4: Experience with a globally-deployed software defined wan. In *SIGCOMM '13*, pages 3–14. ACM, 2013.

[10] G. Zhao, Y.H. Zhao, Y.Q. Chu, Y.P. Jing, and L.C. Deng. LAMOST spectral survey - An overview. *Research in Astronomy and Astrophysics*, 12(7):723–728, 2012.

[11] M.G. Allen, F. Ochsenbein, S. Derriere, T. Boch, P. Fernique, and G. Landais. Extracting photometry measurements from VizieR catalogs. In *Astronomical Society of the Pacific Conference Series*, volume 485, pages 219–228, 2014.

[12] A. Goodman, J. Fay, A. Muench, A. Pepe, P. Udomprasert, and C. Wong. Worldwide telescope in research and education. *arXiv preprint arXiv:1201.1285*, 2012.

[13] Y. Li, L. Zhao, C. Cui, and C. Yu. Fast big data analysis in geo-distributed cloud. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 388–391, Sept 2016.

[14] Zhiming Hu, Baochun Li, and Jun Luo. Flutter: Scheduling tasks closer to data across geo-distributed datacenters. In *Proceedings - IEEE INFOCOM*, pages 423–428, 2016.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[16] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[17] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[19] David A. Papa and Igor L. Markov. Hypergraph partitioning and clustering. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, chapter 61, pages 61:1–61:19. Chapman & Hall/CRC, 2007.

[20] Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[21] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek. Parallel hypergraph partitioning for scientific computing. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pages 1–10, April 2006.

[22] C. J. Alpert, Jen-Hsin Huang, and A. B. Kahng. Multilevel circuit partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):655–667, Aug 1998.

[23] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Papers on Twenty-five Years of Electronic Design Automation*, 25 years of DAC, pages 241–247, New York, NY, USA, 1988. ACM.

[24] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine. The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring. *Scientific Programming*, 20(2):129–150, 2012.

[25] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. SIGCOMM '13, pages 15–26, New York, NY, USA, 2013. ACM.

[26] E. Danna, S. Mandal, and A Singh. A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering. In *IEEE INFOCOM*, pages 846–854, March 2012.

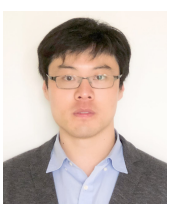
[27] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. EuroSys '10, pages 265–278, New York, 2010. ACM.

[28] S. Dolev, P. Florissi, E. Gudes, S. Sharma, and I. Singer. A survey on geographically distributed big-data processing using mapreduce. *IEEE Transactions on Big Data*, PP(99):1–1, 2017.

[29] P. Li, S. Guo, T. Miyazaki, X. Liao, H. Jin, A. Y. Zomaya, and K. Wang. Traffic-aware geo-distributed big data analytics with predictable job completion time. *IEEE Transactions on Parallel and Distributed Systems*, 28(6):1785–1796, June 2017.

[30] Benjamin Heintz, Abhishek Chandra, and Ramesh K. Sitaraman. Optimizing grouped aggregation in geo-distributed streaming analytics. HPDC '15, pages 133–144, USA, 2015. ACM.

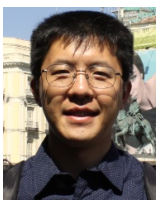
[31] Albert Jonathan, Abhishek Chandra, Jon Weissman, undefined, undefined, undefined, and undefined. Awan: Locality-aware resource manager for geo-distributed data-intensive applications. *2016 IEEE Inter. Conf. on Cloud Engineering (IC2E)*, pages 32–41, 2016.
[32] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low latency geo-distributed data analytics. *SIGCOMM*, 45(4):421–434, 2015.
[33] Chien-Chun Hung, Leana Golubchik, and Minlan Yu. Scheduling jobs across geo-distributed datacenters. SoCC '15, pages 111–124, New York, 2015. ACM.
[34] Jinghui Zhang, Mingjun Wang, Junzhou Luo, Fang Dong, and Junxue Zhang. Towards optimized scheduling for data-intensive scientific workflow in multiple datacenter environment. *Concu. and Comp.: Prac. and Expe.*, 27(18):5606–5622, 2015.
[35] Moritz Steiner, Bob Gaglianello Gaglianello, Vijay Gurbani, Volker Hilt, W.D. Roome, Michael Scharf, and Thomas Voith. Network-aware service placement in a distributed cloud environment. SIGCOMM '12, pages 73–74, USA, 2012. ACM.
[36] M. Selimi, L. Cerd-Alabern, L. Wang, A. Sathiaseelan, L. Veiga, and F. Freitag. Bandwidth-aware service placement in community network micro-clouds. In *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, pages 220–223, 2016.
[37] H. Xu and B. Li. Joint request mapping and response routing for geo-distributed cloud services. In *2013 Proceedings IEEE INFOCOM*, pages 854–862, April 2013.
[38] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang. Joint vm placement and routing for data center traffic engineering. In *2012 Proceedings IEEE INFOCOM*, pages 2876–2880, 2012.
[39] Srinivas Narayana, Wenjie Jiang, Jennifer Rexford, and Mung Chiang. Joint server selection and routing for geo-replicated services. In *Proceedings of the 2013 IEEE/ACM 6th Inter. Conf. on Utility and Cloud Computing*, pages 423–428, 2013.
[40] Arnaud Giersch, Yves Robert, and Frédéric Vivien. Scheduling tasks sharing files on heterogeneous clusters. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 657–660. Springer, 2003.
[41] Arnaud Giersch, Yves Robert, and Frédéric Vivien. Scheduling tasks sharing files from distributed repositories. In *Euro-Par 2004 Parallel Processing*, pages 246–253. Springer, 2004.
[42] Arnaud Giersch, Yves Robert, and Frédéric Vivien. Scheduling tasks sharing files on heterogeneous master–slave platforms. *Journal of Systems Architecture*, 52(2):88–104, 2006.
[43] K. Kaya and C. Aykanat. Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments. *IEEE Transactions on Parallel and Distributed Systems*, 17:883–896, 2006.
[44] K. Kamer, B. Ucar, and C. Aykanat. Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories. *Journal of Parallel and Distributed Computing*, 67:271–285, 2007.
[45] U.V. Catalyurek, K. Kaya, and B. Ucar. Integrated data placement and task assignment for scientific workflows in clouds. In *Proceedings of the fourth international workshop on Data-intensive distributed computing*, pages 45–54. ACM, 2011.
[46] G. Khanna, N. Vydyanathan, T. Kurc, U. Catalyurek, P. Wyckoff, J. Saltz, and P. Sadayappan. A hypergraph partitioning based approach for scheduling of tasks with batch-shared I/O. In *CCGrid*, pages 792–799. IEEE, 2005.
[47] S. Venugopal and R. Buyya. An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids. *Journal of Parallel and Distributed Computing*, 68:471–487, 2008.



**Laiping Zhao** received the B.S. and M.S. degrees from Dalian University of Technology, China in 2007 and 2009, and Ph.D. degree from Department of Informatics, Kyushu University, Japan in 2012. He is currently an associate professor at the School of Computer Software, Tianjin University, China. His research interests include cloud computing and software defined networking.

**Yanan Yang** received the B.S. degree from Yanshan University, China. He is currently pursuing the M.S. degree with the School of Computer Software, Tianjin University. His research interests focus on networking and cloud computing.

**Ali Munir** received the B.S. degree in electronics engineering and the M.S. degree in electrical engineering from the National University of Sciences and Technology, Pakistan. He is currently pursuing the Ph.D. degree with the Computer Science and Engineering Department, Michigan State University. His research interests focus on networking and security.



**Alex X. Liu** received his Ph.D. degree in Computer Science from The University of Texas at Austin in 2006, and is a professor at the Department of Computer Science and Engineering, Michigan State University. He received the IEEE & IFIP William C. Carter Award in 2004, a National Science Foundation CAREER award in 2009, and the Michigan State University Withrow Distinguished Scholar Award in 2011. He has served as an Editor for IEEE/ACM Transactions on Networking, and he is currently an Associate Editor for IEEE Transactions on Dependable and Secure Computing and IEEE Transactions on Mobile Computing, and an Area Editor for Computer Communications. He received Best Paper Awards from ICNP-2012, SRDS-2012, and LISA-2010. His research interests focus on networking and security.

**Yue Li** received the B.S. and M.S. degrees in Computer Science and Technology from Tianjin Polytechnic University and Tianjin University, China, in 2015 and 2017, respectively. He now is an engineer in Meituan-Dianping, the world's largest online and On-demand delivery platform. His research interests include high performance computing and massive data processing.

**Wenyu Qu** received the B.S. and M.S. degrees from the Dalian University of Technology, China, in 1994 and 1997, respectively, and the Ph.D. degree from the Japan Advanced Institute of Science and Technology, Japan, in 2006. She is currently a Professor with the School of Computer Software, Tianjin University. She was a Professor with Dalian Maritime University, China, from 2007 to 2015. She was an Assistant Professor with the Dalian University of Technology, China, from 1997 to 2003. Her research interests include cloud computing, computer networks, and information retrieval. She has authored over 80 technical papers in international journals and conferences. She is on the committee board for a couple of international conferences.