ResLake: Towards Minimum Job Latency and Balanced Resource Utilization in Geo-distributed Job Scheduling

Xinchun Zhang*, Aqsa Kashaf*, Yihan Zou*, Wei Zhang*, Weibo Liao, Haoxiang Song, Jintao Ye,
Yakun Li, Rui Shi, Yong Tian, Wei Feng, Binbin Chen, Zuzhi Chen, Tieying Zhang, Yongping Tang
ByteDance
reslake-paper@bytedance.com

ABSTRACT

At internet scale companies like ByteDance, data is generated and consumed at enormously high speed by many different applications. Achieving low latency on such big data jobs is an important problem. However, the naive approach of aggregating all the data required by a job to a single location is not always feasible in a geo-distributed environment. Similarly, existing approaches in geo-distributed job scheduling often try to minimize WAN usage, which may come at the cost of latency. Another crucial element to ensure low latency is resource load balancing among DCs, which enables flexibility in job scheduling and avoids resource bottlenecks. Therefore, to minimize latency, optimizing job completion time (JCT) while maintaining resource utilization balance is important. To this end, we propose *ResLake*, a global scheduling platform for data-intensive workloads. ResLake aims to reduce JCT of geo-distributed applications while balancing the compute (CPU/Memory) and storage (Disk) usages across DCs and efficiently using WAN interconnections. We have deployed ResLake in ByteDance's production for over 1.5 years. ResLake has scheduled billions of jobs since its deployment. We find that ResLake improves JCT of jobs by at least 20%, and can improve resource utilization balance across DCs by up to 53%.

PVLDB Reference Format:
Xinchun Zhang, Aqsa Kashaf, Yihan Zou, Wei Zhang, Weibo Liao, Haoxiang Song, Jintao Ye, Yakun Li, Rui Shi, Yong Tian, Wei Feng, Binbin Chen, Zuzhi Chen, Tieying Zhang, Yongping Tang. *ResLake*: Towards Minimum Job Latency and Balanced Resource Utilization in Geo-distributed Job Scheduling. PVLDB, 17(12): 3934 - 3946, 2024.
doi:10.14778/3685800.3685817

1 INTRODUCTION

Digital technological advancements have led to the rapid growth and proliferation of data. As a result, internet-scale companies like ByteDance are seeing an increasing growth of data-intensive applications that collect, process, and analyze enormous amounts of data to help them gain useful insights. These internet-scale companies deploy tens of data centers (DCs) across multiple regions in the world to provide low-latency service to their customers. At these geographically distributed sites, data is generated and consumed at an enormously high speed by many different applications. Ensuring low latency for these data-intensive jobs is important to meet service level agreements (SLAs), and to enhance user experience.

Existing big data processing frameworks such as Hadoop [3], Spark [36], Flink [7] and Dryad [17] have been designed to analyze large datasets efficiently. However, all these frameworks assume a single-DC deployment, where network resources are typically uniform and readily accessible, making these solutions infeasible for geo-distributed data analysis. To extend to multiple DCs, a trivial solution would be to aggregate all the data in a single location for processing. However, it is often impractical due to the following reasons. First, the job may have many inputs, whose total data size may be very challenging for storage space in any DC. Second, these data may only be used once, resulting in a waste of resources. Moreover, for high availability, data needs to be replicated and stored at multiple remote DCs to prevent data loss during incidents.

Recent efforts have tried to build on these frameworks to enable data analytics across multiple DCs [14, 28, 30, 31]. However, these frameworks are not optimized for the wide-area network (WAN) bandwidth heterogeneity and limitations [27]. Other works assume that different DCs have uniform computational resources, which does not conform to the reality [16]. Other approaches that perform geo-distributed job scheduling for big data jobs seek to optimize WAN usage [15, 30] as they are designed for a public cloud environment where WAN resources can easily become a bottleneck. However, in a private cloud environment such as ByteDance, network resources are often over-provisioned. Since WAN usage optimization can often come at the cost of latency, these approaches do not directly account for low latency.

Hence, to optimize job completion time (JCT) for geo-distributed big data jobs, we propose ResLake. ResLake is a global scheduling platform for data-intensive workloads, which in addition to reducing JCT, also aims to balance the compute (CPU/memory) and storage (Disk) utilization across DCs and efficiently use the WAN interconnections. We believe that ensuring a balanced resource utilization is crucial for optimizing JCT, as it enables more flexible scheduling and prevents resource bottlenecks that may increase latency. To design ResLake, we formulate the job scheduling problem as a joint optimization problem that minimizes JCT and maximizes resource utilization balance. In the JCT minimization part, we divide the entire scheduling task into meta-tasks, which can be scheduled individually to avoid unnecessary resource blocking. Also, ResLake frequently updates each resource's processing rate to ensure the accuracy of overall approximate JCT. To formulate the resource utilization balance as an optimization problem, we aim to ensure that the resource utilization of each cluster is close to the average cluster utilization. We design ResLake as a layered system. The core

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685817

function of the control layer of ResLake is to perform job scheduling. The compute, storage, and network layers in ResLake support the control layer, and ensure execution of the decisions made by the control layer. Moreover, these layers also provide necessary information to the control layer that aids in scheduling jobs.

ResLake has been deployed in ByteDance's infrastructure for over 1.5 years. We evaluate the effectiveness of ResLake on our production, with an emphasis on achieving its goals of improving average JCT and resource utilization balance across DCs. We present the results for 3 core DCs in ByteDance's infrastructure. We find that ResLake improves the JCT by 20% on average. More than 70% of the jobs show gains in JCT with ResLake, and more than 50% of these jobs show JCT gains of more than 60%. ResLake improves the CPU utilization balance by up to 53% and memory utilization balance by up to 71%. Moreover, as a result of ResLake, the variability in cross-DC read traffic is reduced by 50%. Since its deployment, ResLake has successfully scheduled millions of jobs and has covered almost 100% of big data processing jobs at ByteDance.

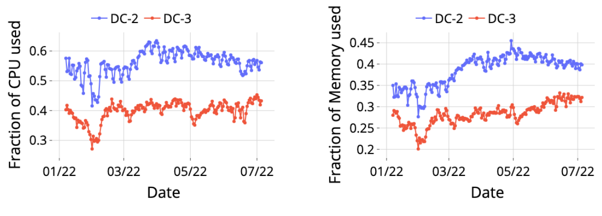
We now summarize the main contributions of this work.

- We formulate the problem of minimizing job completion times in a private cloud environment while maintaining resource balance across DCs.
- We propose an end-to-end system that implements our problem formulation and uses workload characteristics to motivate the design of ResLake.
- We deploy a production-ready system for geo-distributed DCs.
- We show that ResLake improves JCT by over 20% and resource utilization imbalance by over 53%.

Paper Outline The remainder of this paper is organized as follows: Sec. 2 gives the basic motivation behind the design goals of ResLake. Sec. 3 presents a high-level overview of the ResLake infrastructure, and discusses the main design decisions motivated by our workload analysis. Sec. 4 presents the formulation of job scheduling at the control layer of ResLake. Sec. 5 discusses the design of ResLake in detail followed by its implementation in Sec. 6. In Sec. 7, we evaluate ResLake and in Sec. 8 we discuss our related work.

2 MOTIVATION

At ByteDance, majority of submitted jobs are big data jobs, which typically require accessing huge amounts of data of different types. To ensure a good user experience and service level agreement (SLA), it is important to achieve low latency (or completion time) for jobs. In this case, a highly performant scheduling system is needed to manage the scales of jobs and data. For single-DC, in-cluster job scheduling and data placement are well-studied, and there exists known solutions, e.g., YARN [29] or similar technology. However, the problem for geo-distributed setup is still somewhat open, as the design solution is usually company-dependent. In a geo-distributed environment, the scheduling system needs to take various resources into consideration. Existing studies [25] have shown that compute, storage, and network have almost the same probability of becoming performance bottlenecks for data-intensive analysis jobs. As a result, those approaches considering only one or part of the resources are insufficient due to the dependency between job latency and resources. *Therefore, in this work, we propose a solution that directly deals with job latency and resource utilization balance.*



(a) CPU **(b) Memory**

Figure 1: Resource utilization differs by up to 25% across different DCs before the deployment of ResLake.

2.1 ResLake Goals

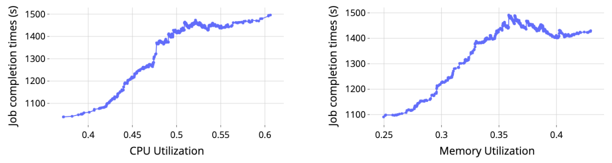
Given the aforementioned context, we now explain the main goals of ResLake.

Reduce Job Completion Times (JCT): JCT is defined as the time it takes to complete a job after it was first submitted by the user. JCT involves the time a job waits for resources, as well as the time spent in fetching data from a remote DC (see Sec. 4.1 for more details). Existing works [15, 26] often try to reduce WAN usage as a proxy for latency. These works mainly consider a public cloud environment where WAN bandwidth is limited and can easily become a bottleneck. However, this is not always true in a private cloud environment for the following reasons:

(1) For self-hosted private clouds, network infrastructure requires an extensive period of planning and construction. Oftentimes there are some degrees of over-provisioning in bandwidth resources;
(2) Network bandwidth is shared by online services and offline jobs. Thus, network traffic exhibits tidal behavior: day-night traffic patterns of online/offline services can be very different. Instead of minimizing WAN usage at all times, we can harvest the peak/off-peak traffic patterns to improve bandwidth utilization and resource efficiency.

Moreover, solely minimizing WAN usage reduces WAN usage for all links irrespective of their available bandwidth, while minimizing JCT may only reduce WAN usage on the bottleneck link of the network. Therefore, ResLake particularly focuses on reducing JCT (which considers WAN latency) instead of reducing WAN usage.

Improve Resource Utilization Balance: While minimizing JCT is important from a user standpoint, it should not come at the cost of skewed resource utilization among DCs/clusters in the system. Fig. 1 shows that significant load imbalance can exist in geo-distributed systems. Skewed resource utilization poses challenges to placing jobs/data at their ideal locations, causing negative impacts on JCT. Fig. 2 illustrates that JCT and CPU/memory resource utilization are positively correlated. There are two degrees of potential imbalance in the system. First, utilization imbalance of the same resource across DCs may result in hot nodes and resource waste. Second, the imbalance across different resources may cause unnecessary blocking due to the dependency among resources. For instance, migrating data closer to a job, or a DC with abundant storage resources, is possible only when there is enough network bandwidth to transfer the data. Hence, ResLake explicitly focuses on balancing load across DCs and among different resources in addition to minimizing JCTs.



(a) CPU **(b) Memory**

Figure 2: JCT is positively correlated with load, e.g., (2a) CPU utilization and (2b) memory utilization.

2.2 Challenges

Achieving the aforementioned goals is not straightforward because of the following aspects:

- **Data Locality Awareness** The naive approach of moving all data required by big-data jobs into one central location can be inefficient and resource-intensive [26, 30]. Therefore, big-data frameworks must prioritize assigning computations to where frequently accessed data resides, a concept known as data locality awareness. However, there can be complex dependencies between data and jobs as we show later in Section 3.2. These dependencies make it hard to achieve data locality.
- **Heterogeneous WAN Bandwidth** Variations in the inter-DC bandwidth can result in unpredictable latency, which can significantly harm the performance of geo-distributed apps [13, 18]. Therefore, it's crucial to manage the Wide Area Network (WAN) bandwidth effectively.
- **Heterogeneous Clusters** There is heterogeneous hardware among clusters and some jobs may require specialized hardware making it important to enforce cluster-aware scheduling. Moreover, the resource capacities at various clusters may also be different. As existing studies [25] have shown that compute, storage, and network have almost the same probability of becoming performance bottlenecks for data-intensive analysis jobs, it is important to consider all resources while scheduling and not just focus on WAN usage.
- **Global Resource Management** While using multiple clusters, users tend to submit jobs to clusters with surplus resources while sacrificing data locality. Moreover, users may need to apply for resources in all clusters, exacerbating the fragmentation of resources. Therefore, a global resource management scheme is needed to balance resource usage while optimizing JCT.

Given the main design goals of ResLake, and the main challenges in designing ResLake, we now present an overview of ResLake in the next section and provide our design details.

3 AN OVERVIEW OF RESLAKE

In this Section, we present the high-level overview of ResLake, including all sub-components in the architecture and the end-to-end workflows in the system. To better facilitate the data-driven analysis, we then discuss some key workload characteristics at ByteDance that motivate our design for ResLake.

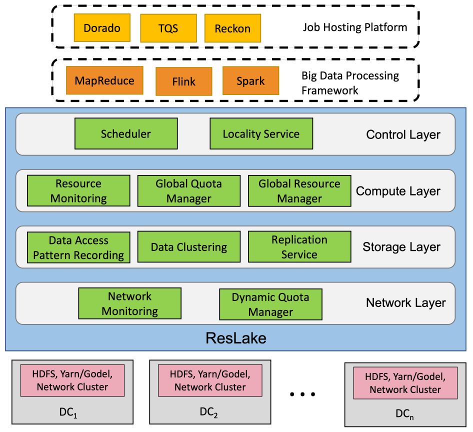


Figure 3: A high-level overview of ResLake's architecture

3.1 High-level System Architecture

We propose ResLake, a centralized scheduling platform for data-intensive applications with a global view of resources and workloads in different DCs/regions. ResLake aims to reduce the job completion time (JCT) of geo-distributed applications while balancing the compute (CPU/ Memory) and storage (Disk) usages across DCs, as well as efficiently use the WAN inter-connecting those DCs. Fig. 3 illustrates the high-level overview of ResLake.

3.1.1 ResLake architecture. ResLake has a layered architecture. We briefly describe each layer and its function below:

Control Layer (ResLake Core): The control layer sits on the top of the ResLake layered architecture, and it is responsible for interacting with the underlying computing layer, storage layer, and network layer. The control layer retrieves comprehensive information regarding computing, storage, and network resources, to make real-time optimal scheduling decisions for job layout. It will also provide guidance and feedback to the other layers to facilitate strategies such as job migration, data migration/replication, and network quota adjustment. In this way, the overall utilization of cross-DC resources can be improved.

Computing Layer: The computing layer provides a unified interface for global computing resource management, virtual queue (i.e., resource group) management, and authentication. In contrast to the in-cluster Resource Manager (RM) in YARN/Godel [29, 34], the Global Resource Manager (GRM) in the computing layer is mainly responsible for managing computing resources across different DCs/clusters. Also, GRM is in charge of maintaining resource statuses in the scheduling process and reporting them back to the ResLake Core for accurate real-time scheduling decisions.

Storage Layer: The storage layer manages data-related features and operations in this architecture. Based on the guidance and feedback from ResLake Core, the storage layer can initiate offline operations, e.g., data replication/migration and possible compression, to enhance the data placement and storage cost savings. Most importantly, the storage layer needs to ensure that frequently accessed data is copied or migrated at the target DCs/clusters before the required deadline for any job that requires the data. On the other hand, the storage layer also reports information (e.g., metadata of the input data) back to ResLake Core for online scheduling.

Network Layer: The network layer is responsible for network conditions monitoring, network quota usage monitoring, and enforcing the network quota assigned by ResLake. On the monitoring front, the network layer reports the latest network conditions and available bandwidth quota to ResLake for job scheduling. On the enforcement front, the network layer needs to ensure the allotted network quota by ResLake is guaranteed, and QoS is satisfied for the given task.

3.1.2 *ResLake end-to-end.*

Based on the high-level context regarding the various components of the ResLake architecture, we now present an end-to-end ResLake overview. We first describe the input and output of ResLake, and then discuss various online and offline mechanisms.

Input: At job submission, users need to specify the information for the job to be scheduled, as well as the required computing quota. Note that only computing quota is needed, and no storage/network quota is specified at this time. In the current design, network bandwidth resource is allocated as a system-level resource pool for ResLake. In ResLake, resource quota is assigned by business application as a whole, and the administrator can further specify fine-grained quotas for specific users. The input can be the database table partition, the file path (optional), or the start offset and length in case of message queues. In ResLake, most jobs' input paths (e.g., Spark SQL, MapReduce) can be automatically parsed. In particular, ResLake explicitly removes the requirements of specifying inputs for any kind of job by analyzing the data access patterns of recurring jobs offline.

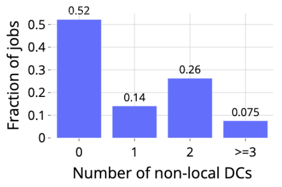
Output: ResLake output consists of a destination DC and cluster where the job will be executed.

Online mechanisms: Some online mechanisms are listed below.

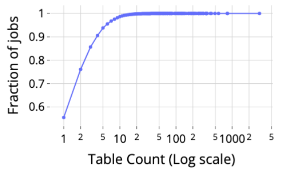
- *Dynamic Job Scheduling:* ResLake is a global service that offers centralized access to compute, storage, and network resources. It can dynamically schedule jobs to geo-distributed DCs/clusters, and make optimal decisions for job placements depending on jobs' input paths.
- *Online Data Synchronization:* For ad-hoc jobs, ResLake can initiate data synchronization on the fly. By using the data cache acceleration capability at the storage layer, the remote read is converted into the local read within the same DC. See more details in Sec. 5.3.
- *Data Access Pattern Recording:* For recurring jobs, ResLake can utilize the analysis results from offline data access patterns mining to improve data locality in scheduling decisions.
- *Dynamic Network Quota Management:* The dynamic quota management recycles unused quota from jobs and assigns additional quota to jobs that have used most of their quota if there is any quota left in the resource pool.

Offline mechanisms: Some offline mechanisms are listed below.

- *Data Replication and Migration:* When a recurring job is submitted, its data synchronization occurs before the job is running. The data access mode is calculated offline by ResLake control layer. Once data replication/migration is determined, the storage layer is notified to synchronize the data to the target DC before the deadline (the specific synchronization timing is determined by the storage layer). To avoid unnecessary storage costs, we will only synchronize data that has an extremely high read-write ratio between DCs.
- *Data Access Pattern Mining:* Related to pattern recording in online mechanisms, ResLake analyzes historical data access characteristics of recurring jobs, generates data access patterns, and combines it with the replica metadata of the storage system.
- *Network Quota Allocation:* When a job is submitted to ResLake, the control layer assigns a bandwidth quota for it. However, ResLake communicates with other in-house network management services and modifies the job's network quota periodically.
- *Path Recommendation for ByteCool* [1]*:* ResLake can analyze the data access behaviors and recommend seldom accessed files to the ByteCool, in order to reduce the storage costs in the company.



(a) Dependency relationship between tables and jobs



(b) Dependency relationship between tables and jobs

Figure 4: (4a) Almost 50% of jobs read data from a non-local DC, and 34% of jobs read data from more than 1 non-local DC. (4b) Almost 50% of the jobs read from more than one table. The distribution is long-tailed, indicating that some jobs read from an arbitrarily large number of tables.

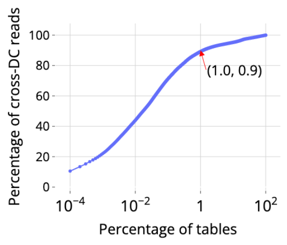
3.2 Workload Driven Design Decisions

Below, we describe some key design decisions in designing ResLake and also provide data-driven analysis to justify our decisions based on the workload characteristics at ByteDance.

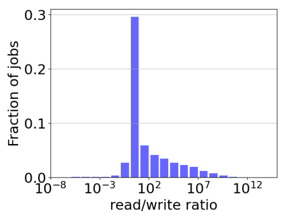
D1: Closely related tables/files are stored in the same DC as much as possible. The dependency relationship between tables and jobs is extremely complex. Currently, there are over 2 million tables in the production system, and over 37% of tables have been active in the past 7 days. Fig. 4b shows the dependencies between tables and jobs. We analyzed the situation of YARN/Godel application accessing data belonging to hive tables and data centers in the recent period. Fig. 4b illustrates that 44.5% of the jobs have access to multiple tables, and 47.5% of the jobs have access to multiple data center data at the same time. To summarize, our analysis results show that *nearly half of the tasks have access to multiple tables or DCs, and the complexity of data dependencies brings great challenges to resource scheduling.*

We analyzed cross-DC network traffic of the recent 936,660 batch jobs and found that the top 10% jobs contribute more than 90% of cross-DC traffic. We also analyzed cross-DC network traffic of the

[1]ByteCool is an in-house data storage system at ByteDance. Erasure Code (EC) [12] encoding is used by ByteCool to achieve data dependability comparable to multiple copies with less redundancy. Consequently, ByteCool can significantly reduce the cost of storage.



(a) Hot Tables



(b) Read-write (RW) ratio

Figure 5: (5a) The top 1% tables contribute more than 90% of cross-DC traffic. (5b) Jobs read more data than write. The majority of jobs (94%) have an RW ratio of more than 1.

top 1 million hive paths and found that the top 1% hive tables contribute more than 90% of cross-DC traffic. *Thus, the job-table relationship follows the 90-10 rule.*

D2: Location of multiple replicas is considered when scheduling jobs. The rationale for multiple replicas is two-fold. First, replication improves data availability and provides the ability for disaster recovery. With multi-replica architecture, one can achieve 3-nines (99.9%) data availability in a year, where the probability of losing a block on a large (4000 nodes) cluster is 0.021% [8]. Second, multiple replicas can improve data locality and reduce WAN usage. Unlike some prior work, there is no concept of "default DC/cluster" for jobs in ResLake, and jobs can be scheduled at any DC if necessary. Intuitively, having data replicas at multiple DCs provides more flexibility for data locality, and helps achieve load balancing among DCs/clusters.

D3: A universal quota and virtual-queue model is needed to support multiple levels (Region/DC/Cluster) of granularity. In practice, each user/application may own multiple physical compute/storage resources (or physical queues) in various DCs/clusters. However, we observed that *the direct interaction between users and physical queues can result in the following issues.* (1) *Poor User Experience:* Users cannot perceive the utilization rate of queue resources and cluster load in real-time, resulting in uneven use of queue resources and even failed job submissions. (2) *Degraded Data Locality:* Users tend to submit jobs to queues with surplus resources, sacrificing data locality and leading to more serious cross-DC read problems. (3) *Resource Fragmentation:* The sub-optimal user decisions may lead to more idle portion of resources (or fragments) that cannot be utilized by the scheduler.

D4: ResLake enforces per-job network quota only for data reading. WAN bandwidth is a scarce resource shared by many applications, and network capacity can oftentimes be impacted by errors and incidents. Thus, it is important to ensure that WAN usage is under close monitoring and proper control. Unlike prior work, instead of minimizing WAN usage, ResLake assigns per-job quota to make sure that WAN does not become a bottleneck. Fig. 5b shows the distribution of read-write (RW) ratios of all jobs in a cluster. *We observed very high RW ratios for the majority of jobs.* Specifically, 94.1% of the jobs have more reads than writes, and the total input size is about 14.2 times of the total output size. Hence, ResLake only enforces network quota on data reads for jobs.

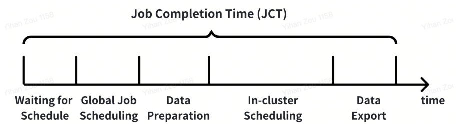


Figure 6: Meta-tasks of a job in ResLake

4 FORMULATION AND MODELING

In this section, we formulate the core decision problem of ResLake, which jointly considers JCT performance and resource utilization in ByteDance's global infrastructure. Acting as a centralized "brain", the main responsibility of the control layer (or ResLake Core) is to coordinate with the underlying computing, storage, and network layers, to achieve the optimal scheduling performance and resource allocation from a global point of view. For ease of explanation in the later part of the analysis, we first list the major decision factors of ResLake Core.

Compute Cost: For a given job, the computing cost includes the CPU, Memory, GPU, and other resources needed, as well as the job's execution time. With the use of jobs' historical running data, this information can be analyzed offline for recurring jobs. For ad-hoc jobs, this information can be estimated based on the input data size.

Storage Cost: For a given job, the storage cost includes its input data size and replica distribution. Such information can be obtained from analyzing the input/output path of the submitted job, and the analysis for majority of the jobs are fully automated.

Network Cost: For a given job, the network cost includes its bandwidth quota, and the WAN bandwidth usage to prepare the job's input data. Such information is crucial for data migration/replication decisions. (See more details about data migration/replication in Section Storage layer.)

No Resource Overbooking: Although one may argue that overbooking can increase resource utilization, it often comes at the cost of severe load imbalance and unnecessary resource competition, causing more issues like unpredictable job performance. Hence, we strictly prohibit resource overbooking in ResLake.

Negligible Intra-DC latency: In reality, compared to cross-DC situation, intra-DC communication usually sees very minimal bandwidth limitation and transmission latency. At ByteDance, the cross-DC network latency is usually 1-2 orders of magnitude higher than intra-DC latency. To simplify the analysis, we focus on the network cost in the context of WAN.

Before diving into details, we first briefly introduce the concept of *meta-task*, which helps the formulation and analysis of the system.

4.1 Meta-tasks

In a geo-distributed system, resources are located in various DCs. Also, skewed resource distribution imposes difficulties to fully utilize resource fragments in a single location. As we discussed in Sec. 2, resource dependency may cause blocking. To avoid unnecessary blocking in scheduling, we use a segmentation-based idea to further divide a job into multiple logical phases that can be scheduled individually. As shown in Fig. 6, the entire pipeline of a job, i.e., from its submission to its completion, can be divided into multiple meta-tasks as follows.

- *Wait For Schedule (WFS).* No resource consumption at this stage.

- *Global Job Scheduling.* ResLake's global scheduling algorithm will dispatch jobs into assigned regions/clusters.
- *Data Preparation.* ResLake will inform the storage layer and network layer to make sure most required data is ready before the job's scheduled time starts.
- *In-cluster Scheduling.* In-cluster scheduling systems (e.g., YARN or Godel) will allocate resources and execute the job.
- *Data Export.* After the job finishes, the output data will be written into the target location.

Note that, the concept of meta-task is related to, but different from the notion of *monotask* in [20]. A monotask is defined as a unit of work that uses only a single type of resource (apart from memory), i.e., CPU, network, or disk. [20] shows that job schedulers can utilize this monotask abstraction to maximize cluster resource utilization. In ResLake, a meta-task is associated with a specific stage in a job's life cycle and has clear relationships with the underlying resource layers. One can view a meta-task as a collection of correlated monotasks in a particular phase of job scheduling.

4.2 Problem Formulation for ResLake Core

The essential task of ResLake is to solve an optimal job scheduling problem. The decision-making process in such a large-scale complex system with various conflicting interests can be formulated as a constrained optimization problem [5, 33]. As mentioned in Section 2, the goal of ResLake is to minimize JCT while simultaneously achieving balanced resource utilization.

4.2.1 JCT Minimization. To do so, we first formulate the JCT minimization (JCT-MIN) problem. Let $\mathcal{M}$ be the set of all meta-tasks for a given job j. From the definition of meta-tasks in the last section, we have

$$t^j_{JCT} = \sum_{m \in \mathcal{M}} t^j_m. \tag{1}$$

For ease of notation, in the following discussion, we will drop the superscript j whenever there is no ambiguity. Let O_m denote the collection of monotasks for the meta-task m. Then, we have

$$t_{JCT} = t_{\text{WFS}} + \sum_{m \in \mathcal{M}, m \neq \text{WFS}} t_m = t_{\text{WFS}} + \sum_{m \in \mathcal{M}, m \neq \text{WFS}} \sum_{o \in O_m} t_o. \tag{2}$$

Since each monotask only uses a single type of resource by definition, we can regroup the above summation by resources, i.e.,

$$t_{JCT} = t_{\text{WFS}} + \sum_{r \in \mathcal{R}} t_r, \tag{3}$$

where $\mathcal{R}$ is the set of all types of resources (e.g., CPU, network, or disk). It is extremely difficult to obtain the exact time duration for a given meta-task, due to factors such as device heterogeneity and random network delay. At a cluster level, for any input data d, we can compute the approximate processing time (APT) that d spent with resource r in cluster c as

$$t^{\text{APT}}_{r,d} = K_d / v_{r,d}(c), \quad v_{r,d} > 0, \tag{4}$$

where K_d is the data size for d and $v_{r,d}(c)$ is the processing rate of resource r for d in cluster c. Note that, $v_{r,d}(c)$ may drift across time due to factors like different load levels or device performance deterioration. Therefore, $v_{r,d}(\cdot)$ needs to be updated at each round to ensure the accuracy of computation. From above, we can obtain the approximate JCT for a given job as

$$\hat{t}_{JCT} = \hat{t}_{\text{WFS}} + \sum_{r \in \mathcal{R}} \hat{t}_r = \hat{t}_{\text{WFS}} + \sum_{d \in \mathcal{D}} \sum_{r \in \mathcal{R}} K_d / v_{r,d}(c), \tag{5}$$

where $\mathcal{D}$ is the set of all required input data for the job. Let $u_c(j)$ be the binary decision variable, such that $u_c(j) = 1$ if and only if job j is assigned to cluster c, and $u_c(j) = 0$ otherwise. Then, the JCT minimization problem for job j can be formulated as

$$\min_{u_c(j)} \quad f_1(u_c(j)) = \sum_{d \in \mathcal{D}_j} \sum_{r \in \mathcal{R}} \frac{K_d}{v_{r,d}(c)} \cdot u_c(j) \tag{6a}$$

$$\text{s.t.} \quad n_{r,c}(j) u_c(j) + \sum_{j' \in \tilde{J}} n_{r,c}(j') \leq N_{r,c}, \quad \forall r \in \mathcal{R}, \forall c \in C, \tag{6b}$$

$$\sum_{c \in C} u_c(j) = 1, \tag{6c}$$

where $\tilde{J}$ is the set of jobs in the system before job j, C is the set of all clusters, $n_{r,c}(j)$ is the amount of resource r that job j is consuming in cluster c, and $N_{r,c}$ is the total amount of resource r in cluster c. In formulation (6), the objective function (6a) minimizes the total approximate JCT for job j. Constraint (6b) is the resource capacity constraint and (6c) ensures that job j will be assigned to a certain cluster. Note that, (6) can be viewed as a greedy approach, where each job tries to find its "locally optimal" scheduling decision, without considering incoming jobs in the future.

4.2.2 Resource Utilization Balance. Next, we consider the resource utilization balancing (RUB) problem. As we discussed in Sec. 2, due to factors like data locality and data dependency, resource utilization imbalance may occur if the scheduling decisions only focus on minimizing JCT. Hence, it is also very important to achieve resource utilization balance among different DCs/clusters in ResLake. For a given resource r in cluster c, the utilization with all jobs $\tilde{J}$ is

$$\eta_{r,c}(\tilde{J}) = \frac{\sum_{j' \in \tilde{J}} n_{r,c}(j')}{N_{r,c}}. \tag{7}$$

Thus, the overall utilization for a resource r in the system is

$$\eta_r(\tilde{J}) = \frac{\sum_{c \in C} \sum_{j' \in \tilde{J}} n_{r,c}(j')}{N_{r,\text{total}}}. \tag{8}$$

To achieve balanced resource utilization, our objective is to have each cluster's resource utilization be close to the average cluster utilization as much as possible, i.e.,

$$\eta_{r,c}(\tilde{J}) \rightarrow \eta_r(\tilde{J}), \forall c \in C.$$

Now, suppose a new job j is submitted. Then, we have

$$\eta_{r,c}(\tilde{J} \cup \{j\}) = \frac{n_{r,c}(j) u_c(j) + \sum_{j' \in \tilde{J}} n_{r,c}(j')}{N_{r,c}}, \tag{9}$$

$$\eta_r(\tilde{J} \cup \{j\}) = \frac{n_r(j) + \sum_{c \in C} \sum_{j' \in \tilde{J}} n_{r,c}(j')}{N_{r,\text{total}}}. \tag{10}$$

To measure the gap between $\eta_{r,c}$ and η_r, we use Mean-Square Error (MSE) as an example. Let $\mathcal{J} = \tilde{J} \cup \{j\}$ RUB problem with MSE can be formulated as

$$\min_{u_c(j)} \quad f_2(u_c(j)) = \sum_{r\in\mathcal{R}}\sum_{c\in C}\left(\eta_{r,c}(\mathcal{J}) - \eta_r(\mathcal{J})\right)^2 \tag{11a}$$

$$\text{s.t.} \quad \eta_{r,c}(\mathcal{J}) \le 1, \quad \forall r \in \mathcal{R}, \forall c \in C. \tag{11b}$$

Note that capacity constraint (11b) is equivalent to constraint (6b) in JCT-MIN problem.

4.2.3 ResLake Core Problem. Now, we present the core problem of ResLake that jointly optimizes JCT and RUB as follows.

$$\begin{aligned} \min_{u_c(j)} \quad & g(u_c(j)) = w_1 f_1(u_c(j)) + w_2 f_2(u_c(j)) \\ \text{s.t.} \quad & \text{constraints (6b) and (6c)} \end{aligned} \tag{12}$$

where w_1 and w_2 is the corresponding weights for JCT-MIN problem and RUB problem. In practice, these weights can be tuned by the ResLake administrator and some privileged users. See more implementation details in Sec. 6. Clearly, to solve (12), ResLake Core layer needs to interact with compute, storage, and network layers to obtain the most updated parameters in the system. On the other hand, once the scheduling decision is made, each underlying layer needs to perform its corresponding functions. Note that we use a weighted-sum approach to scalarize the objective function instead of the Pareto-front-based approach for simplicity [10, 22]. Thanks to the simplified formulation, we can solve (12) by enumerating all possible decisions for our network scale (see our stress test result in Sec 7.3). We will consider the Pareto-front-based solution and other mathematical optimization solvers for future work. We will discuss the distinct designs of these layers in the following section.

5 DESIGN

In this section, we dive into the detailed designs of compute, storage, and network layers, and show how to incorporate the design decisions in Sec. 3.2.

5.1 Computing Layer

Typically, in the absence of global resource management, users apply for resource quota in multiple DCs/clusters separately. This results in several problems. First, with resource quotas in multiple DCs/clusters, users cannot perceive the utilization and cluster load in real time. Second, we observed that users tend to submit jobs to queues with surplus resources, sacrificing data locality. Third, the fragmentation of resources reduces the utilization of total computing resources. Last, in the disaster scenario, resources in the failed cluster cannot be used, resulting in the inability to submit jobs.

To overcome all these aforementioned problems, ResLake proposes a global resource quota management system. This design provides several benefits: 1) It shields business perception from the existence of multiple queues; 2) It solves the problem of uneven resource usage; 3) It improves JCT by considering data locality, and load imbalance; 4) In the event of a disaster, it can solve the problem of migrating queues and recovery operations across data centers and clusters; 5) It also breaks the limitation of queue in a single DC and extends resource sharing from clusters to regions, which contributes to the improvement of global resource utilization.

Quota is the resource quota requested by the user, and the queue is the concretization of quota, which is the resource abstraction provided to the user.

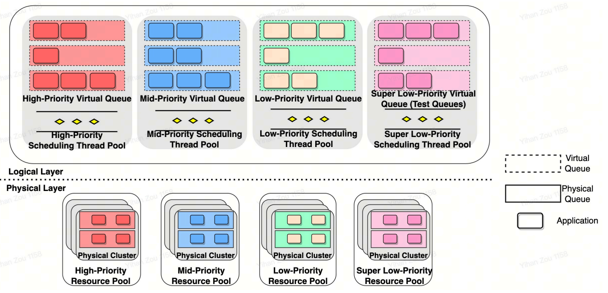

High-Priority Virtual Queue
High-Priority Scheduling Thread Pool
Mid-Priority Virtual Queue
Mid-Priority Scheduling Thread Pool
Low-Priority Virtual Queue
Low-Priority Scheduling Thread Pool
Super Low-Priority Virtual Queue (Test Queues)
Super Low-Priority Scheduling Thread Pool
Logical Layer
Physical Layer
Physical Cluster
High-Priority Resource Pool
Physical Cluster
Mid-Priority Resource Pool
Physical Cluster
Low-Priority Resource Pool
Physical Cluster
Super Low-Priority Resource Pool
Virtual Queue
Physical Queue
Application

Figure 7: Virtual queue management in the compute layer

Global Virtual Queue Management: ResLake introduces the design of global quotas and virtual queues in the computing layer, extending the traditional quota/queue concept from a single cluster to the Region dimension. ResLake virtual queue adopts a three-level tree structure design, which is naturally compatible with Region/DC/Cluster three-dimensions. To ensure the extensibility of queue naming, we define our queues as:

$$\{queue\_namespace\}.\{data\_center\}.\{cluster\_queue\}$$

Among them, *queue_namespace* must be strictly specified, while *data_center* and *cluster_queue* support wildcard ("$*$"). The above queue naming method is compatible with both the cluster-level resource management (in YARN/Godel) and the DC-level federation. To summarize, the hierarchical structure of resource naming is extensible and compatible with queue designs in other systems.
Fault Isolation in Virtual Queues: As the failure domain of ResLake is a resource pool of different SLA levels in different data centers, to achieve the goal of failure domain isolation, we restrict queues of different SLA levels from being merged into virtual queues. That is, virtual queues are divided into high, medium, low, and super-low priorities according to the cluster level. Figure 7 shows how physical computing resources are mapped to virtual queues with different priorities. Each user can join up to N virtual queues of the same priority at the same time. For most scenarios, $N = 2$ is sufficient to separate batch and stream processing. However, we can easily adjust N to meet future needs. There is no limit on the number of existing queues that users can join.
Quota Recycling: The compute layer also exhibits a rebalance service, which periodically scans all virtual queues and executes the rebalance policy. Queue orchestration in the compute layer involves two stages.

(1) Job submission phase: if the job needs to be scheduled away from data in a remote DC, the compute layer will create corresponding physical queues and allocate quotas in the target DC.
(2) Recycling phase: After the job is completed, it will release the corresponding temporary physical queues to prevent resource fragmentation.

To provide more flexibility, the queue rebalance strategy can be customized by users.

5.2 Storage Layer

The storage layer plays a key role in ResLake to optimize JCT. We describe the main functions of the storage layer below:

- Optimize data distribution to ensure that tightly coupled/closely related tables/files are stored in the same data center as much as possible; the storage capacity of each data center and resource consumption are balanced as much as possible.
- Cooperate with ResLake to ensure that recurring and ad-hoc jobs are scheduled to the data-related data center as much as possible; cooperate with the network side to control the rhythm of cross-data center traffic access.
- For hot tables (<10%) provide multi-DC replicas to enable more flexible scheduling and more resources.

We describe each of these aspects below:

Data Distribution Optimization: As mentioned above, placing closely related tables/files in the same datacenter increases the JCT and also reduces the network overhead of remotely accessing data. Therefore, optimizing data distribution is important. However, the problem is challenging because the dependency relationship between tables and jobs is very complex. This also makes it harder to decide which tables can be migrated, and which tables need multi-DC replicas. To solve this problem, ResLake introduces the concept of forming *table clusters*. The tables within the table cluster are tightly coupled with strong dependencies; the dependencies in different table clusters are relatively small. To formulate this relationship, ResLake models the table dependencies as follows. Based on file access traffic data and clustering analysis, this module divides all tables into multiple cohesive table clusters. Then, it builds a weighted dependency graph of a table, where each table is a node in the graph; the weights of the edges between tables are calculated based on the data traffic accessed. The storage system stores the statistical data of all jobs accessing tables.

To illustrate how these weights are calculated, assume job j reads data from Tables T_1 and T_2 to generate Table T_3. When j is running , it reads i_1-sized data from T_1, i_2-sized data from T_2, and writes o_3-sized data to T_3. This module calculates the weight dependencies of the edges (T_1, T_3) and (T_2, T_3) as follows.

$$Weight(T_1, T_3) = i_1 + o_3 * (i_1/(i_1 + i_2))$$

$$Weight(T_2, T_3) = i_2 + o_3 * (i_2/(i_1 + i_2))$$

We select access data from a period of time (usually the past 7 days), traverse all data records, and construct a traffic-weighted dependency graph. To solve this clustering problem, we use mixed integer linear programming (MILP) similar to [11, 35].

Data Access Acceleration: For hot tables or top applications, it is impossible to eliminate cross-DC traffic from a governance perspective. Hence, ResLake implements cross-DC data acceleration to optimize such data access. To support data access acceleration, ResLake implements the following: ResLake sets up an acceleration cluster in each DC to save temporary copies of remote DCs. For recurring jobs, the replication strategy supports periodically copying data from the remote data center to the current data center at the directory partition or file granularity. The strategy supports configuring the lifecycle of the data. This strategy is the main promotion strategy and is suitable for recurring jobs with fixed access patterns and ad-hoc queries for typical partition scans.

5.3 Network Layer

To improve the overall network resource utilization, the control layer of ResLake needs to consider the usage of network resources and ensure the allocation of assigned cross-DC network quota during the job lifecycle. To do this, ResLake implements a network quota manager (NQM) that assigns an initial quota to each job, coordinates with each cluster to ensure the allocation of cross-DC quota, and ensures that the quota is not over-used.

5.3.1 *Network Quota Manager (NQM).*

The network quota manager in ResLake assigns the initial quota to each job, and then dynamically changes the quota if needed. NQM also coordinates the allocation of cross-DC quota for each cluster and ensures that the total quota is not exceeded. We discuss the initial quota allocation strategy and the dynamic quota management strategy below:

Initial Quota Allocation Strategy: The initial quota allocation varies based on the job type. For recurring jobs, NQM mines the input data access pattern in advance according to the historical records, so that an accurate quota estimate can be made for the given job. For ad-hoc jobs, there are two scenarios: default mode and "lazy-read" (cache acceleration for ad-hoc jobs) mode[2].

- *Default Mode:* When the input data path of the job input data is unknown, ResLake will assign default network quota for the job at job scheduling initially. The network quota for the job will be dynamically adjusted later.
- *Lazy Read:* In this case, the input data path for the ad-hoc job can be automatically parsed in advance before the job is submitted. In such a scenario, the initial quota is assigned based on the input data size and the distribution of the data replicas. ResLake can initiate data replication/migration at the storage layer after job submission while allocating the corresponding network quota.

Dynamic Quota Management: The NQM in ResLake interacts with Bandwidth Broker[3] and the in-cluster resource manager where the job is scheduled. For dynamic quota management, the NQM obtains the available cross-DC network quota for each DC from the bandwidth broker. It then coordinates with the cluster-level resource manager (RM) to see the quota usage of clusters and quota requests. The RM monitors the quota usage of jobs, aggregates the information at the cluster level, and provides it to the NQM. For the clusters that need to release quota, the NQM performs quota recycling based on the cluster priority and the available bandwidth. The recovered quota will be included in the total quota resource pool. For the clusters that need to request quota, it executes the following allocation strategy for each priority cluster in order of high and low-priority tasks: When the requested quota is below the available limit of the resource pool, a quota is sequentially allocated to each cluster. When the requested quota exceeds the available quota in the resource pool, a partial quota is allocated to each cluster according to the request ratio, and the remaining part will be allocated in the next round. ResLake centrally adjusts the recycling strategy of each cluster according to the network load. When the network load is low, NQM tries to grant all quota

[2]We note that, although we implemented data cache acceleration for ad-hoc jobs, in practice data may only be accessed once by ad-hoc jobs due to lack of patterns. Thus, this type of data scheduling is highly expensive, and should only be used if necessary.
[3]Bandwidth Broker is a company-wide network service at ByteDance to manage the WAN bandwidth resource and to enforce network quota/QoS.

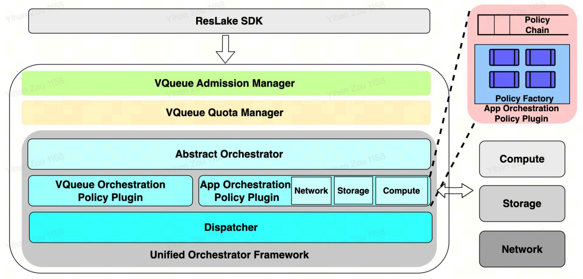


Figure 8: ResLake control layer implementation

requests for each cluster and maximize resource utilization. On the other hand, under high network load, NQM prioritizes high-priority clusters in network quota allocation and forcibly reduces the quota for clusters with lower priority if necessary. To see the details of quota adjustment at cluster and job level, please refer to Section 6.3.

5.3.2 Interaction with External Network Services. BBroker enforces QoS and performs rate-limiting at the application level. The NQM works with BBroker to ensure each job's required bandwidth quota is promptly allocated and effectively utilized. The in-cluster resource manager monitors the quota used by each running job and keeps track of the quota requests. The interaction between ResLake, BBroker, and the in-cluster resource manager is summarized below.

(1) When the job is first submitted, NQM requests a specific amount of cross-DC bandwidth quota from BBroker. Specifically, the job's unique identifier (*classID*), source DC (*srcDC*), and destination DC (*dstDC*) are included in the request body.
(2) Then, BBroker periodically updates NQM with the actual bandwidth usage by the job and the available cross-DC bandwidth resource at the moment.
(3) Next, NQM relays the allotted bandwidth quota to the in-cluster scheduler (YARN/Godel) of the target cluster decided by ResLake. The in-cluster resource manager determines the number of containers to launch, and reports each one's information and the corresponding bandwidth quota back to the BBroker.
(4) Finally, the BBroker ensures that quota is enforced by installing corresponding rules at BBroker agents on the containers.

6 SYSTEM IMPLEMENTATION

In this section, we present some implementation details of ResLake. In our implementation of ResLake, the control and compute layers are intertwined, so we present their implementation together followed by the storage and network layers.

6.1 Control Layer and Compute Layer

The control layer (or ResLake Core) maintains a global view of computing, storage, and network resources to determine the optimal decisions in terms of job latency and global resource balance. We show the architecture of the control layer in Fig. 8. On the top of Fig. 8, the control layer accepts scheduling requests via ResLake SDK or API gateways. First, the submitted requests will go through authentication and permission control by `VQueue Admission Manager` which is implemented by the compute layer. Next, `VQueue Quota Manager` will perform quota management to adjust corresponding quotas based on job information and virtual queues. ResLake implements a Unified Orchestration Framework (UOF), which includes modules such as `VQueue Orchestration Policy Plugin` (or `VQueue Orchestrator`), `App Orchestration Policy Plugin` (or `App Orchestrator`), and `Dispatcher`.

VQueue Arrangement: `VQueue Orchestrator` is responsible for on-demand virtual queue arrangement. When a job is scheduled to a DC without a corresponding registered physical queue, the `VQueue Orchestrator` will apply a temporary physical queue and set up the corresponding quota. When the job is completed, the queue resources will be promptly recycled by ResLake. This allows ResLake to shield the excessive information of physical queues from users. Queue orchestration involves two stages: In the job submission phase, if the job needs to be scheduled away from data in a remote DC, `VQueue Orchestrator` will create corresponding physical queues and allocate quotas in the target DC. In the recycling phase, after the job completes, `VQueue Orchestrator` will release the corresponding temporary physical queues to prevent resource fragmentation.

Dynamic Job Scheduling: `App Orchestrator` is responsible for job scheduling. Essentially, it is the optimization solver for solving the ResLake Core problem (12) in Sec. 4 for job placement. To ensure the performance of the solver output, `App Orchestrator` needs to maintain the most updated view of the computing, storage, and network resources. As shown in Fig. 8, `App Orchestrator` will communicate with underlying compute, storage, and network layers to obtain information including approximate processing rate, CPU/memory utilization, input data location, and network quota, etc. Specifically, by tuning the weights in Eq. (12), `Policy Factory` in `App Orchestrator` can support various standard and customized policies to meet different needs, e.g., to promote data locality or resource utilization balance. The scheduling framework constructs independent thread pools for each priority queue, with the purpose of physical separation of resource pools with different priorities. The separation of job scheduling is conducive to the concurrency of jobs with different priorities, avoiding interference between job schedules with different priorities, and also preventing low-priority task jobs from starving or not being scheduled for a long time. For each queue, to ensure that the scheduler has a global perspective, the jobs of the queue are scheduled serially, but queues with the same priority run concurrently. Each job needs to go through the policy chain to obtain the decision set for the optimal job arrangement.

Scheduling Execution: The `Dispatcher` will ensure the decisions from `App Orchestration` are executed as expected.

6.2 Storage Layer

Storage Metadata Query: The storage layer provides a metadata query interface for file paths, including information such as the datacenter, number of replicas, and size of each replica. After the control layer of ResLake parses the job access paths, it calls this interface to obtain the optimal scheduling data center for the job. Since the file path corresponding to the Hive table partition may contain data in the order of Terabytes, analysis time for this data will be in the order of minutes. Therefore, a metadata warehouse is introduced to reduce the access delay of the path information query interface. 1) Data center information: To avoid information inconsistency caused by data migration, the storage layer periodically refreshes cached data center information. For the newly accessed directories, the path data center information is obtained by sampling the files under the directory and updating the cache. 2) Data size information: the storage system introduces a Data Insight Service, which will periodically dump metadata images into the warehouse and stack directories at all levels of paths to obtain the size information, and finally import ClickHouse to support near real-time queries for data size.

Data Cache Acceleration: To reduce the bandwidth and latency of accessing data across DCs, ResLake analyzes the job's required data and its access behavior offline. Based on the result, the control layer initiates data replication/migration to cache the data at the local DC in advance. The storage system will then forward the client read requests to the data node of the local DC.

DC Relocation: Data insight service collects storage indicators from storage components including name node and data node, and then obtains the current data placement strategy, storage space occupancy, and cross-DC traffic to evaluate the expected resource consumption of data migration.

Data Management service provides data replica relocation capability. The implementation principle is to scan the replica distribution of all file blocks under the directory and determine whether it meets the target distribution strategy. Benefiting from name node replica repair process, the corresponding replicas of the missing data center are supplemented, and the corresponding replica deletion mechanism is used to delete redundant blocks that do not meet the replica placement strategy, ultimately achieving directory-level data center redistribution.

Data Clustering Analysis: Based on historical access characteristics, offline data can be arranged and adjusted more finely and migrated in an orderly manner to achieve the goal of reducing cross-DC traffic. Based on the dependency relationship between computing tasks and data, we construct a Directed Acyclic Graph (DAG) with computing tasks (jobs) and data paths (table or file path) as nodes, and the traffic between query tasks and data paths as edges. The above problem can be summarized as dividing this DAG into several subgraphs so that the sum of edge weights across subgraphs is minimized. The data paths in each subgraph are placed in the same group of DCs, and the computing tasks start from that group of DCs, so the edges across the subgraph are the traffic across the DCs. We use Mixed Integer Linear Programming (MILP) to solve the data clustering problem.

6.3 Network Layer

Initial Quota Allocation: As mentioned in Section 5.3, the network quota manager in the network layer performs initial quota allocation when a job is submitted, and then dynamically manages the quota. For periodic jobs, we assign the initial quota as the average WAN usage of the job in the last n ($n \geq 3$) runs. The NQM uses the API exposed by the storage layer to query the total WAN usage (cross-DC reads). For ad-hoc jobs, we assign a default quota value.

Dynamic Quota Management: For dynamic quota management, we set the recycling policy parameters for each cluster based on the current bandwidth level, including *request_threshold*, *release_threshold*, *increase_step* and *decrease_step*. These parameter values vary based on the current available bandwidth, and the cluster priority, e.g., for a high-priority cluster, when the available bandwidth is high, *request_threshold* is set to 80%. Thus, the high-priority cluster needs to utilize 80% of its current quota to get more quota. Also, *increase_step* is set to 100% for high-priority clusters. So, their current quota will increase by 100%. When the available bandwidth is low, *request_threshold* will increase to 90%, and the *increase_step* will decrease to 50%. The NQM increases the quota by *increase_step* if the current quota usage of a cluster exceeds its *request_threshold*. A similar strategy is used to lower the quota.

Within a given cluster, the in-cluster RM adjusts the individual quota of jobs. When a cluster receives a quota increase, its RM updates the quota of jobs that are more than 80% full for 2 consecutive cycles. If the current network quota of a job is larger than its assigned quota, then the *increase_step* for the job is 100%. When the current network quota is greater than the initial quota, the adjustment is capped by: $min\{0.5 * current\_quota, 0.05 * max\_job\_quota\}$. A similar strategy is used for quota recycling based on the usage of quota per job until the quota of a job reaches *min_job_quota*. The RM aggregates the quota adjustments for all jobs, calculates the total quota adjustment needed at cluster granularity, and gives this data to the NQM. After NQM adjusts the cluster quota based on the request and network conditions, the RM allocates new quotas to the jobs.

7 EVALUATION

In this section, we present the performance evaluation results[4] to demonstrate how ResLake achieves the two main objectives, i.e., reducing JCT and improving resource utilization. We also observe other potential gains in ResLake deployment and analyze how different sub-components can benefit from ResLake.

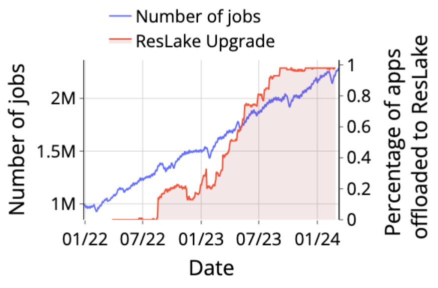
7.1 Experimental Setup

The experimental results in this section were obtained from 3 core DCs in ByteDance's infrastructure. DC-1 and DC-2 are two large DCs with several hundreds of thousands of servers, while DC-3 is a relatively new DC (about 60% scale compared to DC-1) and considerably lower load initially. The levels of load do not necessarily translate to the amount of available resources in these DCs, e.g., DC-1 has the largest number of projects, jobs, and tables, though this does not imply that DC-1 is the largest DC in terms of computing resources. We incrementally deployed ResLake to production through multiple phases. Fig. 9a shows the total number of jobs in the system and the cumulative fraction of jobs covered by ResLake over time. From its release in July 2022, ResLake has covered almost all big data processing jobs in these 3 DCs by September 2023.

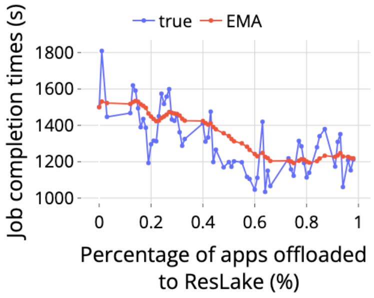
7.2 Main Results

We observe an average improvement of 20% in JCT after the deployment of ResLake. First, we show the results of completion time for all jobs in the above-mentioned 3 DCs to demonstrate ResLake's performance in reducing JCT. Fig. 9b illustrates the average daily JCT in the system and the exponential moving average (EMA) values, concerning the percentage of jobs onboard to ResLake. As ResLake's deployment scale increases, the average

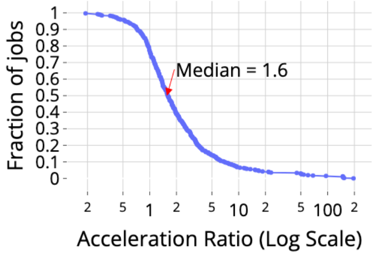
[4]Note that, relative numbers or fractions are reported instead of exact numbers wherever the results contain sensitive business/customer information. The relative numbers, however, are sufficient to show the performance gain of ResLake.



(a) Scales of jobs and ResLake deployment over time

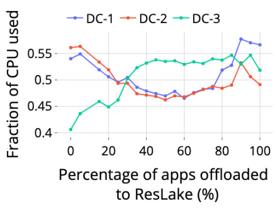


(b) JCT s for jobs against the percentage of jobs offloaded to ResLake

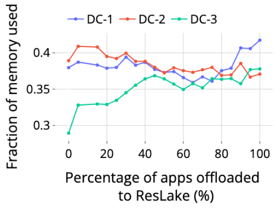


(c) Acceleration ratio ($JCT_{\text{before}}/JCT_{\text{after}}$) of jobs against the fraction of jobs

Figure 9: (9a) Since its deployment, ResLake has successfully scheduled millions of jobs. (9b) As the percentage of apps offloaded to ResLake increases, we observe a reduction in the JCT. (9c) Almost 70% of jobs have an acceleration ratio of more than 1. Almost 50% of the jobs show an improvement in JCT by more than 60%.



(a) CPU



(b) Memory

Figure 10: Overall resource utilization v.s. deployment scale

daily JCT decreases from about 1500 seconds (without ResLake) to about 1200 seconds (at 80% deployment scale), which is a 20% reduction. To give more details, we show the acceleration ratio (defined as the ratio of the JCT before the deployment of ResLake to the JCT after the deployment of ResLake) of recurring jobs in Figure 9c. We see that more than 70% jobs show JCT gains. Moreover, the median acceleration ratio is about 1.6, corresponding to a 60% improvement or higher for 50% of the jobs.

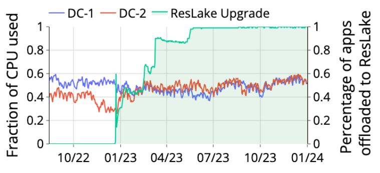
ResLake improves CPU utilization balance by up to 80%, and memory utilization balance by up to 53%. Next, we show how ResLake accomplishes its other main goal of improving resource utilization balance. We plot the average utilization ratios of CPU and memory in each DC w.r.t. the deployment scale of ResLake in Fig. 10a and Fig. 10b, respectively. In Fig. 10a, as the deployment scale increases to 80%, the maximum gap of CPU utilization in 3 DCs decreases from 15% to about 7%, which is a 53% reduction in CPU utilization imbalance. Similarly, in Fig. 10b, we observe that the maximum gap of memory utilization in 3 DCs decreases from 7% to 2%, as the deployment scale increases to 80%, which corresponds to a 71% reduction in memory usage imbalance.

To provide a closer look at ResLake's benefits, we focus on a single cluster with medium priority (MED cluster) in DC-1 and DC-2, and plot its CPU/memory utilization changes as well as the ResLake deployment coverage over time in Fig. 11. Since the MED cluster upgraded to ResLake in Dec 2022, the CPU (Fig. 11a) and memory (Fig. 11b) utilization imbalance between DC-1 and DC-2 have been significantly improved. We also plot the CPU/memory utilization ratios w.r.t. the deployment scale of ResLake in MED cluster in Fig. 11c and Fig. 11d. We observe that the CPU utilization gap between DC-1 and DC-2 decreases from 14% to about 2% which is an 85% improvement. Similarly, the memory utilization gap decreases from 13% to less than 5% which is a 61% improvement.

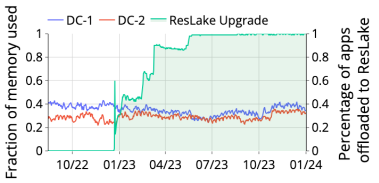
ResLake reduces the variability in cross-DC reads by 50%: To observe the effect of ResLake on WAN usage, we plot the cross-DC read traffic generated by jobs over time, in Figure 12. The boxplot summarizes the data for each month. We also show the percentage of apps offloaded to ResLake in the secondary y-axis to understand the effect of ResLake on cross-DC reads. While we do not observe a significant decrease in the median value of cross-DC reads, we observe a significant change in the spread of values. The unusual increases in Mar-May 2023 are due to a business decision regarding storage decommission in one DC. As a result, lots of data needs to be migrated to other DCs during that time, which is not handled by ResLake. For a fair comparison, we exclude data points in Mar-May 2023 in the following discussion. Before ResLake was deployed, the average cross-DC reads and its peak value are quite diverged, showing a very bursty traffic pattern. As more apps are offloaded to ResLake, the range of values for cross-DC reads is significantly reduced. Overall, by Jan 2024, the inter-quartile range is reduced by half, showing a decrease in the variability of cross-DC reads.

7.3 Other Benefits

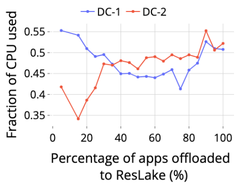
System Scalability: One crucial benefit of ResLake as a centralized control layer for compute, storage, and network resources is to improve the scalability and to eliminate potential bottlenecks. Before ResLake, the job scheduling is handled by in-cluster scheduling services such as YARN and Godel [34]. Pressure test results show that the Resource Manager (RM) of YARN/Godel can process up to several thousands of tasks per minute. With the rapid growth of business demands at ByteDance, a single cluster manager may eventually become the bottleneck of the system soon. In today's production, ResLake is processing several millions of job requests every day as shown in Figure 9a in a single region. Our pressure test result shows that ResLake can easily handle over 30 million job scheduling requests per day. Note that, this number does not reveal the true limit of ResLake system, as the bottleneck is reached at the in-cluster scheduler side with the given number of test clusters.



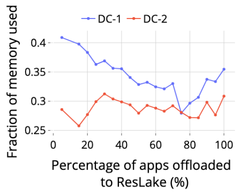
(a) CPU



(b) Memory



(c) CPU



(d) Memory

Figure 11: After the deployment of ResLake, we see significant improvement in CPU utilization balance (11a) and memory utilization balance (11b) in our MED cluster. As the percentage of apps offloaded to ResLake increases, the improvement in CPU utilization balance goes up to 85% (11c), and up to 61% in case of memory (11d).

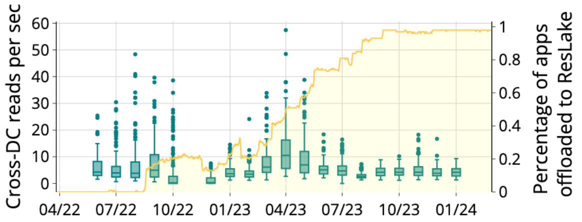


Figure 12: Cross-DC reads per second

From a theoretical analysis, we are confident that the performance limit can be enhanced by increasing the number of processing threads, as ResLake naturally supports parallelism.

Storage Cost Savings by ByteCool: Another great example for ResLake's benefits is to provide useful guidance for the storage layer, to make better decisions on data migration and data compression. To help vacate storage space in some heavily loaded DCs, some less-frequently visited data can be migrated to a remote DC with more abundant storage resources, also referring as "warm/cold data storage". Before ResLake, there were over 1,000 Hive tables (corresponding to multiple EB in data size) stored in ByteCool. By the end of 2023, ResLake helped identify over 10,000 potential tables to move to ByteCool (nearly 20% of them were accepted), which contributes to about 46% of storage savings by ByteCool.

8 RELATED WORK

Many existing efforts try to optimize job scheduling for big data applications in geo-distributed settings [14, 15, 19, 23, 26]. Among these works, many try to optimize for WAN usage as they consider a public cloud environment, where WAN resources can easily become a bottleneck [15, 23]. ResLake on the other hand operates in a private cloud environment and directly optimizes for JCT. Other works ignore the resource utilization balance which can harm JCT [23, 26]. In contrast, ResLake tries to optimize for resource utilization balance actively. Some of these proposed efforts are also limited in scope as they only handle a restricted set of data processing jobs. For example, Iridium only supports MapReduce tasks [26], Geode [23] is limited to handling only recurring SQL jobs. However, ResLake supports a wide range of big data processing frameworks. Other prior efforts [14, 19] that solve the job scheduling problem do not handle data placement, which reduces the application of the system, as it may not always be feasible to move jobs close to data for example, due to hardware requirements of the job. In comparison, ResLake considers the data placement problem in unison with the job scheduling problem to have ultimate flexibility in scheduling jobs to optimize for JCT. Similarly, some existing efforts [14, 19] assume that cross-DC data access for jobs is minimal or non-existent, which does not apply to ByteDance, as shown by our workload patterns in Section 3.2. Other works look at analyzing dependencies and propose dependency-aware scheduling similar to ResLake. Still, they ignore resource utilization balance which is even more important as a result of cross-resource dependencies [6, 21, 24, 32].

Many works present data placement systems to achieve data locality. Akkio [2] provides a data-placement system that optimizes data locality based on jobs, to minimize the data access times. Akkio proposes the concept of micro-shards that allow fine-grained data placement and migration. Similar to Akkio, several other systems manage data locality at shard granularity [4, 9] and do not perform job scheduling. Among systems that only perform data placement, Volley [1] is another example. Volley provides data placement recommendations based on data locality and does not handle the placement leaving it to the users. These works propose a data placement system solely as opposed to ResLake which actively considers job scheduling, and hence they cannot handle cases where it is not feasible or it is costly to move data.

9 CONCLUSION

We presented ResLake, a centralized scheduling platform for geo-distributed data-intensive applications that aims to reduce JCT, while achieving utilization balance across resources and DCs. At its core, ResLake solves an optimization problem to jointly minimize JCT and resource utilization imbalance. The control layer of ResLake coordinates with the underlying compute, storage, and network layer to ensure the effectiveness of various online and offline mechanisms. Extensive evaluation results in ByteDance's production have shown that average JCT and resource utilization balance has improved significantly since the deployment of ResLake. So far, ResLake has covered nearly 100% of big data processing jobs at ByteDance and has been consistently showing its stability and robustness over an extensive period (over 1.5 years). We are excited to share our experience with the community of geo-distributed scheduling systems, and we believe that the solution can benefit a broad audience from both industry and research institutes.

REFERENCES

[1] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and Habinder Bhogan. 2010. Volley: Automated data placement for geo-distributed cloud services. In *NSDI*. USENIX, 2.

[2] Muthukaruppan Annamalai, Kaushik Ravichandran, Harish Srinivas, Igor Zinkovsky, Luning Pan, Tony Savor, David Nagle, and Michael Stumm. 2018. Sharding the shards: managing datastore locality at scale with Akkio. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX, 445–460.

[3] Apache Software Foundation. [n. d.]. *Hadoop*. https://hadoop.apache.org

[4] Masoud Saeida Ardekani and Douglas B Terry. 2014. A {Self-Configurable}{Geo-Replicated} Cloud Storage System. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX, 367–381.

[5] Dimitri Bertsekas, Angelia Nedic, and Asuman Ozdaglar. 2003. *Convex analysis and optimization*. Vol. 1. Athena Scientific.

[6] Marcel Blöcher, Lin Wang, Patrick Eugster, and Max Schmidt. 2021. Switches for HIRE: resource scheduling for data center in-network computing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 268–285. https://doi.org/10.1145/3445814.3446760

[7] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering* 38, 4 (2015), 28–38.

[8] Robert J. Chansler. 2012. Data Availability and Durability with the Hadoop Distributed File System. *login Usenix Mag.* 37 (2012). https://api.semanticscholar.org/CorpusID:2146015

[9] Carlo Curino, Evan Philip Charles Jones, Yang Zhang, and Samuel R Madden. 2010. Schism: a workload-driven approach to database replication and partitioning. (2010), 48–57.

[10] Kalyan Deb. 2001. *Multiobjective Optimization Using Evolutionary Algorithms. Wiley, New York.* Wiley.

[11] Chris HQ Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D Simon. 2001. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings 2001 IEEE international conference on data mining*. IEEE, IEEE, 107–114.

[12] IBM Documentation. 2024. IBM Storage Ceph – Edge clusters, Erasure-coding. https://www.ibm.com/docs/en/storage-ceph/7?topic=components-erasure-coding

[13] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. 15–26.

[14] Zhiming Hu, Baochun Li, and Jun Luo. 2016. Flutter: Scheduling tasks closer to data across geo-distributed datacenters. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, IEEE, 1–9.

[15] Yuzhen Huang, Yingjie Shi, Zheng Zhong, Yihui Feng, James Cheng, Jiwei Li, Haochuan Fan, Chao Li, Tao Guan, and Jingren Zhou. 2019. Yugong: Geo-distributed data and job placement at scale. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2155–2169.

[16] Chien-Chun Hung, Ganesh Ananthanarayanan, Leana Golubchik, Minlan Yu, and Mingyang Zhang. 2018. Wide-area analytics with multiple resources. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 1–16.

[17] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (Lisbon, Portugal) *(EuroSys '07)*. Association for Computing Machinery, New York, NY, USA, 59–72.

[18] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 3–14.

[19] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. 2015. Network-aware scheduling for data-parallel jobs: Plan when you can. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 407–420.

[20] Tatiana Jin, Zhenkun Cai, Boyang Li, Chengguang Zheng, Guanxian Jiang, and James Cheng. 2020. Improving resource utilization by timely fine-grained scheduling. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. ACM, New York, NY, USA, Article 20, 16 pages.

[21] Aqsa Kashaf, Vyas Sekar, and Yuvraj Agarwal. 2020. Analyzing third party service dependencies in modern web services: Have we learned from the mirai-dyn incident?. In *Proceedings of the ACM Internet Measurement Conference*. 634–647.

[22] Il Yong Kim and Olivier de Weck. 2006. Adaptive weighted sum method for multiobjective optimization: A new method for Pareto front generation. *Structural and Multidisciplinary Optimization* 31 (02 2006), 105–116. https://doi.org/10.1007/s00158-005-0557-6

[23] Peng Li, Song Guo, Toshiaki Miyazaki, Xiaofei Liao, Hai Jin, Albert Y Zomaya, and Kun Wang. 2016. Traffic-aware geo-distributed big data analytics with predictable job completion time. *IEEE Transactions on Parallel and Distributed Systems* 28, 6 (2016), 1785–1796.

[24] Jinwei Liu and Haiying Shen. 2016. Dependency-Aware and Resource-Efficient Scheduling for Heterogeneous Jobs in Clouds. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 110–117. https://doi.org/10.1109/CloudCom.2016.0032

[25] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. 2015. Making sense of performance in data analytics frameworks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (Oakland, CA) *(NSDI'15)*. USENIX Association, USA, 293–307.

[26] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 421–434.

[27] Srikanth Sundaresan, Walter De Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. 2011. Broadband internet performance: a view from the gateway. *ACM SIGCOMM computer communication review* 41, 4 (2011), 134–145.

[28] Radu Tudoran, Gabriel Antoniu, and Luc Bouge. 2013. Sage: geo-distributed streaming data analysis in clouds. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE, IEEE, 2278–2281.

[29] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing* (Santa Clara, California) *(SOCC '13)*. ACM, New York, NY, USA, Article 5, 16 pages.

[30] Ashish Vulimiri, Carlo Curino, Philip Brighten Godfrey, Thomas Jungblut, Konstantinos Karanasos, Jitendra Padhye, and George Varghese. 2015. Wanalytics: Geo-distributed analytics for a data intensive world. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. ACM, 1087–1092.

[31] Lizhe Wang, Jie Tao, Rajiv Ranjan, Holger Marten, Achim Streit, Jingying Chen, and Dan Chen. 2013. G-Hadoop: MapReduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems* 29, 3 (2013), 739–750.

[32] Shaoqi Wang, Wei Chen, Xiaobo Zhou, Liqiang Zhang, and Yin Wang. 2019. Dependency-Aware Network Adaptive Scheduling of Data-Intensive Parallel Jobs. *IEEE Transactions on Parallel and Distributed Systems* 30, 3 (2019), 515–529. https://doi.org/10.1109/TPDS.2018.2866993

[33] Laurence A Wolsey and George L Nemhauser. 2014. *Integer and combinatorial optimization*. John Wiley & Sons.

[34] Wu Xiang, Yakun Li, Yuquan Ren, Fan Jiang, Chaohui Xin, Varun Gupta, Chao Xiang, Xinyi Song, Meng Liu, Bing Li, Kaiyang Shao, Chen Xu, Wei Shao, Yuqi Fu, Wilson Wang, Cong Xu, Wei Xu, Caixue Lin, Rui Shi, and Yuming Liang. 2023. Gödel: Unified Large-Scale Resource Management and Scheduling at ByteDance. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA,) *(SoCC '23)*. ACM, New York, NY, USA, 308–323.

[35] Zhengxi Yang, Zhipeng Jiang, Wenguo Yang, and Suixiang Gao. 2023. Balanced graph partitioning based on mixed 0-1 linear programming and iteration vertex relocation algorithm. *Journal of Combinatorial Optimization* 45, 5 (2023), 121.

[36] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (oct 2016), 56–65.