Yugong: Geo-Distributed Data and Job Placement at Scale

Yuzhen Huang[12*†], Yingjie Shi[2*], Zheng Zhong[12†], Yihui Feng[2], James Cheng[1†], Jiwei Li[2], Haochuan Fan[2], Chao Li[2], Tao Guan[2], Jingren Zhou[2]
[1]The Chinese University of Hong Kong
[2]Alibaba Group
[1]{yzhuang, zzhong, jcheng}@cse.cuhk.edu.hk
[2]{yuzhen.hyz, yingjie.shi, zhongzheng.zz, yihui.feng, jiwei.ljw, haochuan.fan, li.chao, tony.guan, jingren.zhou}@alibaba-inc.com

ABSTRACT

Companies like Alibaba operate tens of data centers (DCs) across geographically distributed locations. These DCs collectively provide the storage space and computing power for the company, storing EBs of data and serving millions of batch analytics jobs every day. In Alibaba, as our businesses grow, there are more and more cross-DC dependencies caused by jobs reading data from remote DCs. Consequently, the precious wide area network bandwidth becomes a major bottleneck for operating geo-distributed DCs at scale. In this paper, we present Yugong — a system that manages data placement and job placement in Alibaba's geo-distributed DCs, with the objective to minimize cross-DC bandwidth usage. Yugong uses three methods, namely project placement, table replication, and job outsourcing, to address the issues of high bandwidth consumption across the DCs. We give the details of Yugong's design and implementation for the three methods, and describe how it cooperates with other systems (e.g., Alibaba's big data analytics platform and cluster scheduler) to improve the productivity of the DCs. We also report comprehensive performance evaluation results, which validate the design of Yugong and show that significant reduction in cross-DC bandwidth usage has been achieved.

PVLDB Reference Format:
Yuzhen Huang, Yingjie Shi, Zheng Zhong, Yihui Feng, James Cheng, Jiwei Li, Haochuan Fan, Chao Li, Tao Guan, Jingren Zhou. Yugong: Geo-Distributed Data and Job Placement at Scale. *PVLDB*, 12(12): 2155-2169, 2019.
DOI: https://doi.org/10.14778/3352063.3352132

1. INTRODUCTION

Big companies and cloud providers today manage tens of geographically distributed data centers (DCs) across the globe [20, 2, 40]. A typical DC consists tens of thousands of physical machines [53, 61, 52, 48]. These DCs provide the computing and storage capacity for many Internet-scale businesses. When managing geo-distributed DCs at such scale, the precious cross-DC bandwidth becomes a major bottleneck [56, 22, 32, 49].

In Alibaba [1], a large scale data management and analytics platform, called MaxCompute [3], manages tens of DCs in geographically distributed locations. Each of these DCs consists of thousands to tens of thousands of servers, and the DCs are inter-connected by wide area networks (WANs). Collectively these DCs store 5 million tables and execute 7 million analytics jobs every day for various business applications (e.g., Taobao [2], Tmall [3], etc.) in Alibaba. Jobs in MaxCompute form complex *data dependencies* through the production and consumption of massive amounts of data each day. While most of the dependencies are within a local DC, as our businesses grow, dependencies from remote DCs are increasing rapidly. On a typical day, jobs in MaxCompute incur hundreds of PBs of data transfer through WANs due to cross-DC dependencies.

The WANs connecting the geo-distributed DCs, however, have low capacity [43, 54, 18]. The WAN bandwidth is in the order of Tbps, while in comparison the aggregated bandwidth for intra-DC networks can be considered arbitrarily large. The WAN latency is 10-100 times higher than the intra-DC latency in Alibaba. The WANs are also much more expensive compared with the intra-DC networks [9, 17]. Moreover, the capacity growth rate of the WANs is significantly slower compared with the growth of our business application demands (similar findings have also been reported in [56, 55, 18]). Over time, the cross-DC bandwidth has become a very precious resource and also the performance bottleneck for MaxCompute's operations. The WAN cost (before the deployment of Yugong) took up a significant portion of the overall operating cost of MaxCompute — a great financial burden given the massive scale of MaxCompute's daily operations.

The cross-DC bandwidth problem is not uniquely caused by Alibaba's internal business applications, but is common in any enterprise that provides globally deployed services (e.g., all kinds of Web services, mobile APPs, E-commerce), which are normally run in geo-distributed DCs [15, 12, 31, 8, 41, 13]. There are also many big data analytics and database management systems such as Hive [50], Spark [60], Flink [11], etc., which may run in one DC but the data could be transferred from remote DCs via a transparent layer such as a platform like MaxCompute. In fact, MaxCompute itself is a PaaS solution for Alibaba Cloud [4] to transparently manage all underlying (intra-DC and cross-DC) data movements for both Alibaba's internal customers (i.e., various business units such

*Co-first-authors ordered alphabetically.
†This work was done when the authors were in Alibaba.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352132

[1]https://www.alibaba.com/
[2]https://www.taobao.com/
[3]https://www.tmall.com/
[4]https://www.alibabacloud.com/

as Taobao and Tmall) and its Cloud customers, so that customers may use existing systems for their applications on the Cloud without caring the complicated optimizations for cross-DC data management.

The above discussion motivates us to develop **Yugong** — *a system that aims to reduce cross-DC bandwidth usage (internally for MaxCompute but extensible as a more general solution for data/project placement in geo-distributed DCs)*. Reducing cross-DC bandwidth usage is a challenging problem. Recent research on geo-distributed analytics, e.g., [43, 54], only optimize the bandwidth usage for a small number of jobs and focus on the bandwidth reduction for each individual job by scheduling tasks within a job across multiple DCs. However, modern DCs usually manage thousands of projects (a project is a group of logically related data and jobs for a specific business application) and execute millions of jobs for these projects each day, There are also complex dependencies among jobs, tables of projects and DCs. Existing solutions were not designed for such large scale, complex production environments. In addition, the loads and the availability of resources (e.g., CPU, memory, disk, network bandwidth) change dynamically in different DCs, which aggravates the problem of data/project placement in geo-distributed DCs.

In the Yugong project, we consider a more holistic solution. We conduct extensive analysis to understand the projects and their tables, the daily jobs and their data dependencies in Alibaba's geo-distributed DCs. Based on the valuable insights obtained from the analysis, we divide the problem into an offline component and an online component. We formulate the offline component as two separate problems, **project placement** and **table replication**, and the online component as the problem of **job outsourcing**. Project placement manages the placement of new projects (e.g., a new business, or divisions of an existing business) and the migration of existing projects among the DCs. Table replication replicates (hot) partitions of tables to multiple DCs and determines the life span of these replicas. Job outsourcing dynamically schedules certain jobs out of their original DCs to other DCs. The three problems address different aspects of bandwidth usage across our DCs as follows. Project placement re-groups projects with high dependencies with each other together in one DC. Table replication reduces cross-DC bandwidth usage by moving data closer to jobs, while job outsourcing achieves the goal by considering the resource availability and loads of the DCs and moving jobs to where data are.

The solutions to these three problems are implemented as the corresponding modules in the Yugong system to provide services to MaxCompute to reduce cross-DC bandwidth usage. After the deployment of Yugong, significant reduction on cross-DC bandwidth usage has been observed compared with the previous method used in our production. On average, the bandwidth usage reduction is about 70-80% of the total cross-DC dependencies, and the amounts of incoming cross-DC bandwidth saving for different DCs range from several PBs to tens of PBs each day. Yugong has also improved load balancing among the DCs with its job outsourcing service, which supplements MaxCompute's geo-distributed job scheduling capability.

Our main contributions are summarized as follows:

- An in-depth analysis of very large scale production workloads consisting of thousands of projects, millions of jobs and tables, and their dependencies across geo-distributed DCs in Alibaba.
- An analytical model that formulates the problem of minimizing cross-DC bandwidth usage, an effective strategy that decouples the problem into simpler problems, and efficient algorithms that are devised based on the insights obtained from our workload analysis.
- A production-ready implementation of our solution for large scale geo-distributed DCs.
- A comprehensive performance study to validate the effectiveness of our approach and Yugong's performance.

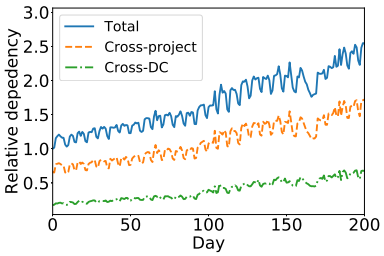


Figure 1: Trend of dependency growth

Paper outline. The remainder of this paper is organized as follows. Section 2 gives the basic settings of the problem studied in this paper. Section 3 presents the workload analysis. Section 4 gives an overview of Yugong. Section 5 discusses the detailed problem formulation and solutions. Section 6 describes implementation details. Section 7 reports the performance evaluation results. Section 8 discusses related work. Section 9 summarizes our work.

2. BACKGROUND

We first describe the basic settings necessary for our discussion in the subsequent sections. Yugong was developed to serve the geo-distributed DCs managed by MaxCompute [3], which is a general purpose, fully fledged, multi-tenancy data processing platform developed at Alibaba Group from ground up since 2009. MaxCompute provides data warehouse solutions at very large scale.

User data in MaxCompute are organized by **tables**, which are in turn grouped into **projects** according to business functionality (e.g., search services for Taobao form a project). Most of these tables are *read-only* [5] **time-partitioned tables** and in many applications *a partition is created for each table each day*. For example, the sales data of Taobao are kept as a time-partitioned table and the daily sales records are appended to this table every day.

Each computation workload submitted by a user is called a **job**. *Each job belongs to only one project* as it is usually launched by the project group and uses the project's resource quota for its execution. By default, each project along with its jobs and tables is assigned to a **default DC**. Each job runs in only one DC because each DC hosts thousands to tens of thousands of machines and has enough resources to process even the largest jobs within a reasonable time. We also expect that the computing and storage power within a DC grows at a similar pace as the tables and jobs grow.

Jobs in MaxCompute form complex data **dependencies** as jobs may read tables from other projects that are in different DCs, which leads to cross-DC communication. Initially, the concept of project is to group logically related jobs and tables in one DC to reduce cross-DC communication. However, as businesses grow in Alibaba, modularization and the reusability of the modules are emphasized and thus the interactions among projects are getting more

[5]Yugong (and MaxCompute) also supports updatable tables, as long as tables support snapshot reads and the snapshots to be accessed can be determined at query compilation time.

Table 1: Project statistics

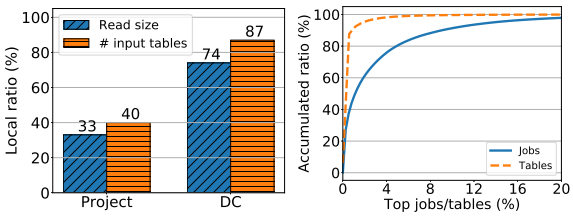
	median	70ile	90ile	95ile
Job count	0.03618	0.14341	1.28655	3.38794
Table count	0.03332	0.14381	1.19097	3.25267
Partition count	0.00490	0.07122	1.32603	3.38401
Physical size	0.00037	0.01144	0.47186	2.06538
CPU	0.01646	0.13485	1.44358	4.14529
Mem	0.01372	0.10932	1.37686	4.09082

Table 2: Job statistics

	median	70ile	90ile	95ile
Input count	0.26685	0.80056	26.4000	77.6000
Output count	0.26685	0.26685	0.26685	0.80056
Input size	0.00029	0.01415	1.50871	8.95521
Output size	0.00001	0.00046	0.05425	0.32171
CPU	0.00136	0.00876	0.21882	1.25955
Mem	0.00055	0.00409	0.12494	0.97678

Table 3: Table and partition statistics

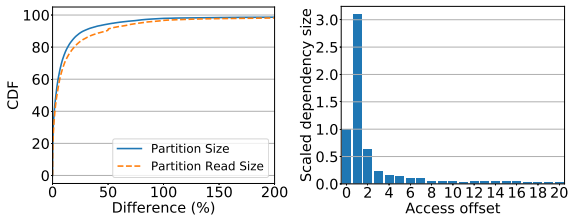
	median	70ile	90ile	95ile
Table size	0.10853	4.85782	236.730	988.152
Partition size	0.00099	0.01481	0.24773	0.95386



(a) Local dependency ratio at the project/DC level

(b) Accumulated dependency ratio

Figure 2: Dependency analysis



(a) Partition/read size difference

(b) Cross-DC dependency size

Figure 3: Table and partition access patterns

complex, resulting in significantly increased cross-DC bandwidth usage. Figure 1 plots the trend of dependency growth among our DCs over 200 consecutive days in 2018. The dependencies among our DCs increased significantly over the last year. *Specifically, the cross-DC dependencies increased around 3x during the period. Thus, reducing the cross-DC bandwidth usage becomes critical.*

3. WORKLOAD ANALYSIS

We conducted a comprehensive analysis on the workloads of Alibaba's geo-distributed DCs to explore (1) how the workloads use cross-DC bandwidth and (2) how we can reduce the bandwidth usage. We present some of our important findings in this section. The findings were obtained based on 7 of our DCs unless otherwise stated.

3.1 Projects, Jobs and Tables

We first present some statistics of the projects, jobs, tables and partitions managed by MaxCompute in Tables 1-3. The numbers in these tables are ***normalized*** [6] by their *mean* values. For example, the *mean* value of the number of daily jobs in the projects is approximately 3200, and thus the *median* job count in Table 1 is about $(0.03618 \times 3200) \approx 116$.

From Table 1, we can see that most projects are small projects, i.e., a 70ile project is only 0.1x of the *mean* in terms of job/table counts and CPU/memory usage, while the *median* values are even much smaller. However, there are also a small number of large projects, i.e., a 95ile project has around ten thousand of daily jobs and several thousand of tables, occupying several petabytes of the physical storage. In fact, *various statistics of projects, jobs, tables and partitions in Tables 1-3 follows a power-law distribution*.

In Table 2, the entries of the input count/size are normalized by the *mean* values of the output count/size, which shows that *jobs normally have more input partitions than output partitions, and their input sizes are much larger than their output sizes*. In Table 3, the entries of the table size are normalized by the *mean* values of the partition size, which shows that *a partition of a table is normally orders of magnitude smaller than the entire table*.

[6]We admit that the exact numbers would be easier to understand, but unfortunately cannot be reported in the paper as they contain sensitive business/customer information. However, we emphasize that the normalized numbers (and other approximate numbers) presented in this paper are sufficient for drawing the conclusions and findings that are necessary for our problem formulation and solutions.

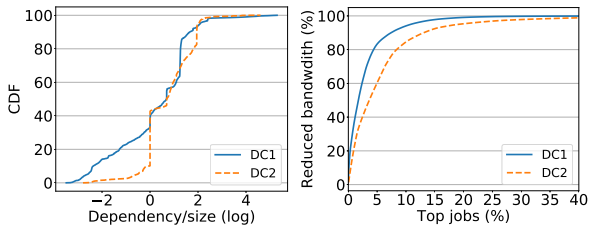
3.2 Dependency Analysis

Next we report the dependency analysis results. Figure 2a gives the local dependency percentage at both the project level and the DC level on a typical day, where *local dependency* refers to data read by a job from tables belonging to its own project or from tables in the local DC. Only 33% of the data and 40% of the input tables read by jobs are from their own projects. About 74% of the input data are from local DCs, while 87% of the input tables are from local DCs. The results show that *there is a large amount of cross-DC dependencies, i.e., 26% of the total dependency size is cross-DC, which amounts to hundreds of PBs of data transfer through WANs per day*.

Figure 2b plots the accumulated dependency ratio for the top jobs and tables ranked by their cross-DC dependencies. The top 5% of the jobs result in 80% of the total dependencies and the top 0.57% of the tables account for 80% of the total dependencies. The results suggest that *a small number of hot tables and large jobs contribute to a significant amount of the overall cross-DC dependencies*.

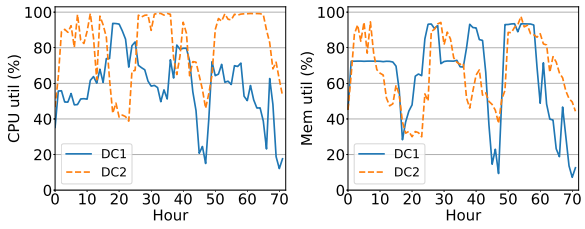
3.3 Table Access Pattern Analysis

The partitions of tables are usually created and accessed by *daily recurring jobs, which consist of more than 76% of the SQL jobs in MaxCompute*. We first analyze the recurrent patterns of the partitions, as well as their access patterns. We compare the sizes of any two consecutive partitions, and plot the CDF of their absolute difference (calculated as $||P_{i+1}| - |P_i||/|P_i|$) as the solid curve in Figure 3a. We also plot the absolute difference between the total read sizes of any two consecutive partitions (i.e., the total amount of data read from each partition by all jobs) on two consecutive days as the dotted curve in Figure 3a. The size differences between the consecutive partitions are small, e.g., 80% of the consecutive partitions have a difference less than 10%, while the differences



(a) |dependency|/|partition| (b) Top-k% dependency jobs

Figure 4: Potential gain of scheduling tables and jobs



(a) CPU utilization (b) Memory utilization

Figure 5: Resource utilization patterns

between the total read sizes of any consecutive partitions are just slightly larger. *In general, the partition size remains rather stable over time and the partition access patterns are recurrent daily.*

Jobs may not just read the most recent partitions of a table. If a job reads a partition that was created k days ago, we regard the *access offset* for the job reading the table as k. *Most of the daily recurring jobs have fixed table access pattern*, i.e., different instances of a job launching on different days read tables with the same access offset. We analyze the access offset of the time-partitioned tables, by aggregating the cross-DC dependency size for all jobs in a day by their access offsets as shown in Figure 3b. The result suggests that *recent partitions are more frequently accessed, especially those with access offsets 0 and 1. Besides, the dependency size decreases exponentially as the access offset increases.*

3.4 Potential Gain

Next we investigate the amount of partitions that are worth replicating to a specific remote DC. If the total amount of dependencies from a remote DC to a local partition (i.e., the total amount of data that jobs in a remote DC need to read from the local partition) is larger than the size of the local partition, replicating this partition to the remote DC can save cross-DC bandwidth. Figure 4a plots the CDF of the ratios of the total remote dependency size to the size of each corresponding partition to be replicated to two of our DCs. We omit partitions with 0 dependency size from these two DCs. The result shows that *more than half of the partitions have remote dependency size larger than the partition size (indicated by the portion with log ratio greater than 0), meaning that replicating these partitions instead of reading them remotely can reduce the total cross-DC bandwidth usage*.

Figure 4b reports the potential gain achieved by scheduling out the top k% of the jobs that have the largest dependencies from other DCs. The figure shows that *scheduling out a small percentage of jobs can reduce a significant amount of the remote dependencies and hence the cross-DC bandwidth usage.*

3.5 Resource Utilization

Figure 5 plots the CPU and memory utilization of two DCs over 72 hours. *The resource utilization patterns are rather dynamic and unpredictable.* Even though there are many recurring jobs, their start time and completion time can vary considerably on different days because a recurring job can only be launched when all its dependencies are resolved and its completion time can be affected by the resource utilization situation. In addition, there are also ad hoc jobs that are difficult to predict their submission time and completion time. *Different DCs also have different resource bottlenecks in different time periods.* For example, DC1 is bottlenecked at memory from Hours 50 to 60, while DC2 is bottlenecked at CPU during that time period.

3.6 Summary of Workload Analysis

We summarize our findings from the workload analysis as follows:

- **[F1]** Projects, jobs and tables follow power-law distributions. The cross-DC dependencies among jobs and tables exhibit the long-tail phenomenon. A relatively small number of hot tables and large jobs contribute to a large portion of the total cross-DC dependencies.
- **[F2]** The inputs of jobs are much larger than their outputs in terms of both data size and number of partitions.
- **[F3]** Partitions of a table created on consecutive days have similar size. The size of a partition is hundreds of times smaller than that of its table. Most jobs are recurrent and the table access patterns on consecutive days are stable.
- **[F4]** Recent partitions of tables are more frequently accessed. The amount of dependencies decreases exponentially as the access offset increases.
- **[F5]** Some tables are frequently read by jobs in remote DCs and thus replicating these tables can save cross-DC bandwidth. Also, scheduling some jobs to run in their non-default DCs can reduce cross-DC bandwidth usage.
- **[F6]** The DCs have dynamic and unpredictable resource utilization patterns, and they may have different resource bottlenecks during the same time period.

4. AN OVERVIEW OF THE YUGONG APPROACH

Based on the findings of our workload analysis, we propose three methods to reduce the cross-DC bandwidth usage: **project placement**, **table replication**, and **job outsourcing**. We give an overview of these three methods below, while we present the detailed problem formulation and solutions in Section 5.

Project Placement. Project placement includes (1) the placement of new projects in DCs and (2) the migration of existing projects from DCs to DCs. In the ideal case, highly related projects are grouped in the same DC such that the total cross-DC bandwidth is minimized. The optimal project placement problem is a variant of the bin packing problem, which is NP-hard. In fact, even if an optimal solution can be found, practically it is still infeasible for the following reasons. In MaxCompute we have already hosted thousands of projects with up to hundreds of PBs of data associated with each project, and thus a re-placement of all projects among the DCs would incur an extremely large amount of bandwidth usage, which is not affordable in our production environment as the DCs

Table 4: Notations

Notation	Description
$L_i(d)$	The life span of a replica of table i in DC d
$p(i), p(j)$	The project that table i or job j belongs to
$T(p), J(p)$	The set of tables or jobs that belong to project p
tp_i^t	The time partition of table i created on day t
S_i	The size of a partition of table i
$R_i^t(j)$	The amount of data read from partition tp_i^t by job j
$R_i^t(p)$	The amount of data read from tp_i^t by $J(p)$
$R_i^t(d)$	The amount of data read from tp_i^t by $J(p)$, $\forall p$ in DC d
$X_{p,d}$	$X_{p,d} = 1$ if project p is assigned to DC d; 0 otherwise
$BW(d)$	The total incoming cross-DC bandwidth for DC d
$BW^{opt}(d)$	The optimal $BW(d)$ given unlimited storage budget
$S^{rep}(d)$	The storage size for replicas at DC d

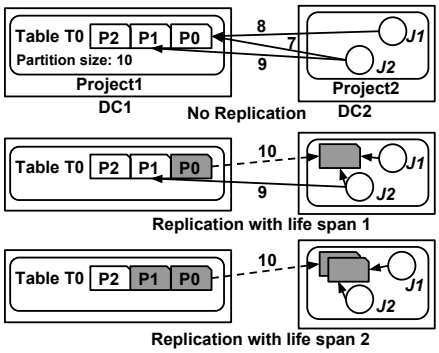


Figure 6: An example of table replication with life span

are being heavily used and our normal business operations must not be interrupted at any time. In addition, even if we have an optimal placement of all projects currently, over time the current placement would no longer be optimal because projects (e.g., the number of jobs, the resource consumption of jobs, table sizes, etc.) and their dependencies may change as our businesses keep developing.

For the above reasons, we change the optimal project placement problem into a project migration problem, which only changes the placement of a small number of projects incrementally (based on F1, Section 3.6). Yugong monitors the cross-DC bandwidth usage under the current project placement and periodically assesses whether a (small, affordable) change of project placement can have a significant reduction on the bandwidth usage. It then executes the new project migration plan.

Table Replication. Some table partitions are frequently read by jobs from remote DCs and thus replicating these partitions to the remote DCs helps reduce the cross-DC bandwidth usage (according to F5). We may also cache partitions for more than one day as jobs may read old partitions. However, there is a trade-off because storing the replicas longer results in less bandwidth usage but uses more storage space. Assume that all tables are time-partitioned and the table access patterns are similar on consecutive days (according to F3), the problem becomes to decide which table partitions are worth replicating (and to which DCs) each day and how long their replicas should be cached under a storage constraint. The objective is to minimize the total cross-DC communication each day, which includes the bandwidth used for sending the replicas to remote DCs and the bandwidth consumed by remote reads in the case when the required partitions are not replicated.

Job Outsourcing. Outsourcing a job, i.e., scheduling a job out of its default DC, to run in a remote DC where some of its input tables are stored may reduce the cross-DC bandwidth usage (according to F5), especially when the input size is large. This may also improve the overall resource utilization (hence save the production cost) by balancing the loads and the utilization of various resources (e.g., CPU, memory, disk, network) among the DCs (based on F6).

5. PROBLEM FORMULATION AND SOLUTIONS

Approach Summary. We divide our approach into an offline component and an online component. Both *project placement* and *table replication* are done offline; while *job outsourcing* requires an online solution since the scheduling decision needs to consider the loads and resource utilization of the DCs. We first present an analytical model to describe the offline component, which is then separated into two individual problems: a *project migration problem* and a *table replication problem*. For project migration, we consider an unlimited storage budget for the replicas. Then, we develop a heuristic that decides the life span of the replicas given a fixed storage budget. The online component schedules jobs to their non-default DCs according to both the static job information (e.g., the cross-DC bandwidth cost when scheduling a job to run in a remote DC and the predicted resource utilization) and the dynamic cluster/network information (e.g., available resources, quota, etc.).

Notations. Table 4 lists the frequently used notations. Most of these notations will be used directly in our subsequent discussions and readers may refer to Table 4 for their definitions.

Life Span of Table Replicas. We first introduce the notion of *life span* for table replicas, as motivated by the fact (i.e., F4 in Section 3.6) that the recent partitions of tables are more frequently accessed. We define the life span $L_i(d)$ for a replica of table i in DC d as the number of most recent partitions that should be stored for table i in DC d. Let t_{cur} be the current day (or the time interval of the current partition of a table, if the time unit of a partition is not a day). In other words, DC d stores the replica of partitions tp_i^t of table i, where $t_{cur} - L_i(d) < t \leq t_{cur}$. For example, if the life span of the replicas for a table in a DC is 2 days, then on Day 7, the partitions of the table that were created on Day 6 and Day 7 are replicated and stored in that DC. On Day 8, the partition that was created on Day 6 becomes stale, and is thus deleted and replaced by the current partition that was created on Day 8. With the notion of life span, all reads to the partitions that are covered by the life span are local (as they are replicated in the local DC). The cost to maintain the life span consists of (1) the cross-DC bandwidth cost required each day to replicate the latest partition to the remote DC, i.e., S_i; and (2) the storage cost for storing the most recent $L_i(d)$ replicated partitions in the remote DC, i.e., $S_i \times L_i(d)$. Thus, the trade-off is that a longer life span turns more remote dependencies into local reads but increases the maintenance cost. Apparently, setting the life span to infinity (i.e., replicating all partitions of tables) minimizes the bandwidth, but is impractical as the cost is too high.

We illustrate the benefit of table replication with life span in Figure 6. P0, P1 and P2 are time partitions of table T0 that were created today, yesterday and two days ago, respectively. T0 belongs to Project 1 in DC1. Assume that the size of each partition is 10 units, and jobs in DC2 require $(7 + 8)$ units of data from P0 and 9 units from P1. If jobs in DC2 directly read the partitions from DC1, the total cross-DC bandwidth cost is 24 units. If the life span is 1 for T0 in DC2, P0 is replicated in DC2 and the reads to P0 becomes local. Each day, we need to replicate the latest partition with a cross-DC replication bandwidth cost of 10 units to maintain the

life span. Thus, the total cross-DC bandwidth cost is 19 units and the replication storage size is 10 units (for storing one partition). If the life span is 2 for T0 in DC2, then the storage cost is 20 units and the cross-DC bandwidth is 10 units each day (by replicating the latest partition), making all reads local. We do not need to consider a longer life span for T0 in DC2 in this example because there is no job in DC2 reading a partition of T0 that is two days old (i.e., P2 is not read).

The above example may look simple. But the challenge is that to determine which table partitions we should replicate (or cache) and to which remote DCs, and find the optimal life span (i.e., how long should we cache) for each table in each remote DC, in order to *minimize cross-DC bandwidth usage under the condition that we have limited replication storage size and in MaxCompute we have millions of tables (each with hundreds of time-partitions) and millions of daily jobs.*

5.1 Analytical Model

Assumptions. In the following context, we attempt to minimize the total cross-DC bandwidth usage in a day. We assume that the WANs connecting each pair of DCs have the same unit cost. We assume (based on F3) that the sizes of all partitions from the same time-partitioned table are the same, jobs are recurring daily and their table access patterns are the same on consecutive days. Thus, we only need to consider jobs running on one day. We use "all jobs" to refer to "all jobs running on the current day t_{cur}". The table access patterns are generated by jobs running on t_{cur} only. The jobs and access patterns are actually changing slowly over days and we will discuss how we can adapt to changes in Section 5.3.

Model Formulation. We first elaborate $R_i^t(p)$ and $R_i^t(d)$ in Table 4. $R_i^t(p)$ is the total amount of data read from partition tp_i^t by all jobs of project p, given as follows.

$$R_i^t(p) = \sum_{j \in J(p)} R_i^t(j). \tag{1}$$

$R_i^t(d)$ is the total amount of data read from partition tp_i^t by all jobs of all projects in DC d, given as follows.

$$R_i^t(d) = \sum_p R_i^t(p) X_{p,d}. \tag{2}$$

We also denote $R(d)$ as the *table access matrix* of DC d, i.e., $R(d)[i,t] = R_i^t(d)$ for all table i and time t.

Given a project placement plan $\mathcal{P}$, a table replication plan $\mathcal{R}$ (i.e., the life span L for all tables in DC d), the total cross-DC bandwidth cost for d each day can be calculated by the sum of (1) the total cost of all the remote reads for all the partitions that are not covered by $\mathcal{R}$ and (2) the total cost for replicating remote partitions to DC d.

$$BW(d) = \sum_{i:X_{p(i),d}=0} \Big(\sum_{l \geq L_i(d)} R_i^{t_{cur}-l}(d) + S_i \times sgn(L_i(d)) \Big), \tag{3}$$

where $sgn(x) = 1$ when $x > 0$ and $sgn(x) = 0$ when $x = 0$. The total storage size needed to maintain the replicas is given by

$$S^{rep}(d) = \sum_i S_i \times L_i(d). \tag{4}$$

Constraints. Every project should belong to exactly one DC:

$$\forall p, \sum_d X_{p,d} = 1. \tag{5}$$

Let cpu_p, mem_p, sto_p be the CPU, memory and storage consumption for project p, respectively, and CPU_d, MEM_d and STO_d be the CPU, memory and storage limit of DC d. Projects in a DC should satisfy the following resource constraints:

$$\forall d : \sum_p cpu_p X_{p,d} \leq CPU_d, \tag{6}$$

$$\forall d : \sum_p mem_p X_{p,d} \leq MEM_d, \tag{7}$$

$$\forall d : \sum_p sto_p X_{p,d} \leq STO_d. \tag{8}$$

The storage required for storing the replica should not exceed a replication storage budget STO_d^{rep}:

$$\forall d : S^{rep}(d) \leq STO_d^{rep}. \tag{9}$$

As we explained in Section 4, in MaxCompute we want to limit the negative impact caused by project re-placement and thus we only select a small, affordable number of projects to be migrated. We denote the maximum number of projects that we want to migrate in a phase as $MigCount$, and $DC_{orig}(p)$ be the original DC that project p belongs to. Then, after project migration takes place, we have:

$$\sum_p (1 - X_{p,DC_{orig}(p)}) \leq MigCount. \tag{10}$$

We also consider some project-DC constraints. For example, if project p can only be placed in its original DC, we have $X_{p,DC_{orig}(p)} = 1$ at all time and $X_{p,d} = 0, \forall d \neq DC_{orig}(p)$. If project p cannot be moved to some DC d, we have $X_{p,d} = 0$.

Objective. The objective is to find the project placement plan $\mathcal{P}$ and the table replication plan $\mathcal{R}$ such that the total cross-DC bandwidth cost for each day is minimized, while satisfying all the constraints:

$$\min \sum_d BW(d). \tag{11}$$

Analysis. This analytical model is computationally intractable as it is an integer programming problem with $(|T| + |P|) \times |DC|$ integer variables, where $|T|$ is the total number of tables, $|P|$ is the total number of projects, and $|DC|$ is the number of data centers.

We make use of our findings, as summarized in Section 3.6, to simplify the problem. Since jobs tend to read the recent partitions and the size of their dependencies on partitions decreases exponentially as the access offset increases (i.e., F4), it is reasonable to first assume that we have sufficient/unlimited replication storage size for project migration and then use a heuristic to decide the life span under a fixed storage budget. This simplification greatly reduces the complexity of the model as we essentially remove Constraint (9).

With this simplification, we can decouple the problem into two problems: (1) a project migration problem that first finds the project placement plan $\mathcal{P}$ assuming unlimited replication storage budget, and (2) a table replication problem that finds the table replication plan $\mathcal{R}$ given $\mathcal{P}$, while satisfying Constraint (9). The decoupling is also natural to our production environment because project placement/migration cannot be executed frequently due to the higher migration cost, while the table replication plan can be updated far more frequently.

5.2 Project Migration

In the simplified project migration model, we remove Constraint (9). Given a project placement plan, the minimum cross-DC bandwidth cost $BW^{opt}(d)$ for DC d can be obtained by: (1) reading all

required partitions of each table i from the DC of table i, if the total amount of data read from table i is smaller than the total size of its partitions; or (2) otherwise storing all the partitions of table i in DC d and replicating its latest partition each day. Thus, $BW^{opt}(d)$ is given by:

$$BW^{opt}(d) = \sum_{i:X_{p(i),d}=0} min\{\sum_t R_i^t(d), S_i\}. \quad (12)$$

The above summation over $R_i^t(d)$ along time dimension t helps remove the complexity of jointly considering the replication strategies for every time partition. Our objective becomes finding a project placement plan $\mathcal{P}$ such that the overall BW^{opt} is minimized:

$$\min \sum_d BW^{opt}(d). \quad (13)$$

Even with this simplification, the number of projects and tables remain large as we have thousands of projects and millions of tables. We make use of the power-law distribution of the cross-DC dependencies (i.e., F1) to further reduce the problem size. As shown in Figure 2b, a small number of tables contribute to a large portion of the cross-DC dependencies. Thus, we only consider the top tables that have the largest dependency size in our project migration model. Besides, when solving the problem, we also found that migrating a small number of projects can significantly improve our current project placement, while the improvement degrades rapidly with further migration as the projects that have great influence are already placed in a suitable DC.

Our project migration strategy can also be used to solve the new project placement problem. We first place the new projects in DCs according to the loads of the DCs, and then solve the project migration problem by setting $MigCount$ to the number of new projects.

5.3 Table Replication

Given the project placement plan $\mathcal{P}$ computed in Section 5.2, we then find a table replication plan (i.e., find the life span $L_i(d)$ for each table i in each DC d) to minimize the total cross-DC bandwidth cost, while satisfying Constraint (9). We first assume (based on F3) that the table access patterns are the same on consecutive days, and devise a heuristic for the life span for all tables in different DCs. Then we remove this assumption and consider the dynamic maintenance of the life span for the table replicas, as the table access patterns are actually changing gradually over time. Note that the table access matrix $R_i^t(d)$ in our solutions include only tables that do not belong to projects in DC d, i.e., $X_{p(i),d} = 0$, since we only care remote reads. For simplicity, we omit DC d in the subsequent discussion.

A DP Solution. The optimal table replication plan under a given replication storage budget and a given project placement plan can be obtained through dynamic programming (DP). We denote $dp(i, s)$ as the minimum cross-DC bandwidth cost that can be obtained by considering the first i tables under the replication storage size constraint s. Let the life span for table i be L_i. The storage size for storing the replica for table i is $L_i \times S_i$. Similar to Equation (3), the cross-DC bandwidth cost incurred for reading partitions of table i is the sum of (1) the total cost of all remote reads for the partitions that are not covered by the life span and (2) the replication cost: $BW_{L_i} = \sum_{l \geq L_i} R_i^{t_{cur}-l} + S_i \times sgn(L_i)$.

The DP transition function is given as follows:

$$dp(i, s) = \min_{L_i}\{dp(i-1, s - L_i \times S_i) + BW_{L_i}\}. \quad (14)$$

Algorithm 1: k-probe life spanning

Input : Table set T, table access matrix R_i^t, partition size S_i, replication storage budget STO^{rep}, and probing variable k
Output: The life span L_i for each table i

```
maxPQ ← ∅ // a max-heap with Gain as key
for i ∈ T do
    L_i ← 0
    for k' in 1 to k do
        Calculate Gain_i^{k'} by Equation (15)
        maxPQ ← {Gain_i^{k'}, (i, L_i)} if Gain_i^{k'} > 0
used ← 0
while maxPQ ≠ ∅ do
    {Gain_i^{k'}, (i, l_i)} ← maxPQ.dequeue
    if l_i = L_i and used + S_i × k' ≤ STO^{rep} then
        used ← used + S_i × k'
        L_i ← L_i + k'
        for k' in 1 to k do
            Calculate Gain_i^{k'} by Equation (15)
            maxPQ ← {Gain_i^{k'}, (i, L_i)} if Gain_i^{k'} > 0
```

The minimum cross-DC bandwidth cost for the first i tables with storage budget s is to enumerate all possible life spans L_i for table i and take the minimum of them.

The time complexity of the DP solution is $O(|T||L||storage|)$, where $|T|$ is the number of tables (in the order of tens of thousands to millions), $|L|$ is the possible life spans for each table (normally a few hundred), and $|storage|$ is the number of replication storage units used in the DP formulation (in the order of thousands of millions: PBs of storage budget divided by MBs of partition size). Thus, the DP solution is too expensive.

K-Probe Greedy Heuristic. As an alternative to the DP algorithm, we propose an efficient greedy algorithm. At each step, the algorithm advances the current life span L_i for table i by up to k units, which is greedily determined by a marginal gain defined as follow:

$$Gain_i^k = \begin{cases} (\sum_{0 \leq t < k} R_i^{t_{cur}-L_i-t} - S_i)/(S_i \times k) & L_i = 0 \\ \sum_{0 \leq t < k} R_i^{t_{cur}-L_i-t}/(S_i \times k) & L_i > 0 \end{cases} \quad (15)$$

Intuitively, the nominator is the total cross-DC bandwidth cost that can be saved by advancing L_i by k units, while the denominator is the storage space needed to store the extra k partition replica. If $Gain_i^k > 0$, it means that replicating the extra k partitions can further save cross-DC bandwidth. Note that we need to subtract S_i from $Gain_i^k$ when $L_i = 0$ because we need to use cross-DC bandwidth to replicate partition $tp_i^{t_{cur}}$, while for $L_i > 0$ this cost is already covered by the case $L_i = 0$.

Algorithm 1 presents the greedy algorithm. The algorithm first initializes a max-priority-queue (maxPQ) to keep all possible $Gain_i^k$, if $Gain_i^k > 0$, for $L_i = 0$ for each table i. Then it keeps dequeuing the maximum $Gain$ from the maxPQ until the queue becomes empty. Suppose that the current maximum $Gain$ is $Gain_i^{k'}$ for a certain table i, we advance L_i by k' units if the storage budget still allows for another k' replicas. The "$l_i = L_i$" condition is to make sure that for all $Gain_i^{k'}$ calculated based on the current L_i, where $0 \leq k' < k$, only one k' will be used to advance L_i. After L_i is advanced, new gains $Gain_i^{k'}$ will be calculated based on the updated L_i and enqueued in the maxPQ.

The time complexity of Algorithm 1 is $O(k|T||L|\log(k|T||L|))$, which is significantly smaller than that of the DP algorithm since k is only in the order of hundreds. We show that the greedy algorithm

obtains the optimal bandwidth cost given sufficient storage budget as follows.

Theorem 1. *Setting STO^{rep} to the actual replication storage size used when the optimal bandwidth cost in Equation (12) is achieved and k to be the maximum life span, Algorithm 1 computes a table replication plan that gives the same optimal bandwidth cost as Equation (12).*

We give our proof sketch as follows. Let L_i^{opt} be the life span for table i when the optimal bandwidth cost in Equation (12) is achieved. Consider the current life span L_i for table i in Algorithm 1. We have $L_i < L_i^{opt}$, and the gain for advancing L_i to L_i^{opt} is higher than to any $l > L_i^{opt}$, since L_i^{opt} requires less storage while incurring the same amount of reads (note that L_i^{opt} is obtained given unlimited replication storage budget and so it will replicate any partition as long as the remote read size from that partition is larger than the partition size, as given in Equation 12). Thus, if our storage budget is the same as the actual replication storage size used to obtain the optimal bandwidth cost in Equation (12), then the gain for advancing L_i to L_i^{opt} will be dequeued from the maxPQ at some point. And after that, the gain for further advancing L_i becomes 0 and will not be enqueued.

Incremental Maintenance. Our greedy solution only considers fixed table access patterns for now. In practice, the table access patterns are actually changing (though slowly) over time due to the growth of business and occasional ad-hoc jobs. Thus, we need to update the table replication plan periodically. Assume that the plan is updated every δ days. The problem now becomes, given a current replication plan $\mathcal{R}$, we need to find a new replication plan $\mathcal{R}'$ such that (1) it can serve as a good replication plan for the coming δ days and (2) the bandwidth cost for transiting from $\mathcal{R}$ to $\mathcal{R}'$ is minimized. As an example for the transition cost, suppose the life span for table i is L_i in $\mathcal{R}$ and L_i' in $\mathcal{R}'$, and $L_i < L_i'$, meaning that $\mathcal{R}'$ covers more partitions for table i than $\mathcal{R}$. Thus, extra bandwidth is needed to replicate the older partitions from $(t_{cur} - L_i')$ to $(t_{cur} - L_i)$ in order to transit from $\mathcal{R}$ to $\mathcal{R}'$.

The simplest way to update the replication plan is to rerun Algorithm 1 every δ days but it may incur a significant transition cost. Considering the cost incurred by replicating older partitions, we propose a simple modification to the gain function. Let $I_{i,t}$ be an indicator such that $I_{i,t} = 1$ if tp_i^t is covered by $\mathcal{R}$ and $I_{i,t} = 0$ otherwise. We define G_i^t as the amount of bandwidth that can be saved if the new plan covers tp_i^t.

$$G_i^t = R_i^t - (1 - I_{i,t}) \times S_i/\delta \tag{16}$$

The intuition is that if a partition tp_i^t is not in the replication plan, then including it in the plan needs to pay for a *penalty* that amounts to the amortized bandwidth cost for replicating tp_i^t over the δ days. That is, we do not encourage replicating older partitions unless the gain is substantially large. By replacing R_i^t with G_i^t in Equation 15, we obtain a new gain function:

$$Gain_i^k = \begin{cases} (\sum_{0 \le t < k} G_i^{t_{cur} - L_i - t} - S_i)/(S_i \times k) & L_i = 0 \\ \sum_{0 \le t < k} G_i^{t_{cur} - L_i - t}/(S_i \times k) & L_i > 0 \end{cases} \tag{17}$$

Apart from using a new gain function, we also take the average of the table access matrix R_i^t over the previous δ days to reduce the effect of the oscillations in the table access patterns.

5.4 Online Job Outsourcing

Outsourcing jobs to remote DCs that store the input data can be beneficial if the input data size is large (according to F5). However, we need to consider if the remote DCs have free resources to run the jobs and if the expected job completion time (including waiting time in the remote DCs) is shorter than that in the default DC of a job. Job outsourcing can also be utilized to balance the loads among DCs (according to F6). Thus, the outsourcing decision is made online as it requires dynamic online cluster information.

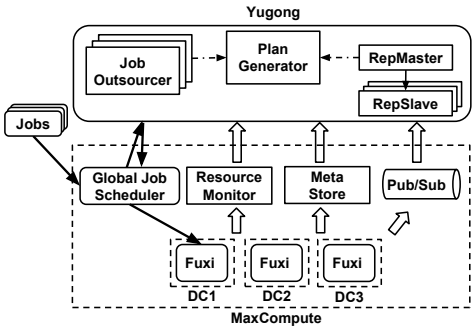


Figure 7: Yugong Components

By default, jobs are launched in their project's default DC as we only allocate resource quota for each project in its default DC. To enable job outsourcing, we create another quota group in each DC for launching jobs from remote DCs and thus we only outsource a limited number of jobs to their non-default DCs.

Job outsourcing mainly considers reducing the cross-DC bandwidth cost while at the same time taking the loads of the DCs into account. We use a simple score function to make the decision of outsourcing job j to DC d:

$$Score(j,d) = \alpha \times \frac{1}{Cost(j,d)} + \beta \times \frac{AvailResrc(d)}{WaitT(j,d)}, \tag{18}$$

where $Cost(j,d)$ and $WaitT(j,d)$ are the total cross-DC bandwidth costs and the estimated waiting time if we outsource job j to DC d, and $AvailResrc(d)$ is the amount of available resources in DC d. Note that $Cost(j,d)$ includes sending all necessary information/data to DC d for the job execution and transferring back the job output back to the default DC. We carefully tune the parameters α and β to favor a small cross-DC bandwidth cost.

6. SYSTEM IMPLEMENTATION

In this section, we provide some implementation details of Yugong. The architecture of Yugong is shown in Figure 7. Yugong is directly plugged into MaxCompute to provide services for project placement/migration, table replication and job outsourcing. We may also connect Yugong with other DC management systems in a similar way.

6.1 Plan Generator

The Plan Generator in Yugong is responsible for generating project placement plans and table replication plans. It takes in the cross-DC dependency information from a MetaStore, which is a key-value store that keeps all the static historical information of jobs, tables and projects in MaxCompute. Then it employs an optimization solver to solve the project migration problem and provides project migration recommendation. We use a home-brewed solver developed by the Intelligent Decision Lab inside Alibaba [16], which is built specifically for large scale constrained programming problems. The Plan Generator also creates new table replication plans periodically using the k-probe life spanning algorithm. The replication storage budget is set for each DC individually based on the

storage usage of each DC. The Plan Generator runs the k-probe algorithm with different update intervals to find the best update interval for incremental maintenance of the replication plan.

The Plan Generator keeps monitoring the workload statistics and bandwidth usage of the DCs, and generates a new project migration plan when there is a significant change in the workloads or when the bandwidth usage becomes worse than a threshold. Meanwhile, the table replication plan is updated periodically according to the best update interval. Projects are only migrated infrequently due to the large migration cost, while the table replication plan is updated more frequently.

6.2 Replication Service

After the project placement/migration plan and the table replication plan are generated, their execution is actually converted into many replication jobs. To execute the placement/migration plan, the partitions of the tables in the projects are replicated to the destination DCs. Before the migration of a project completes, jobs are still launched in the original DC of the project. After the migration completes, Yugong deletes the project's tables from the original DC and updates the MetaStore about the new location of the project. To execute the table replication plan, existing partitions covered by the replication plan are replicated to the destination DCs, while the replication of new partitions will be triggered online when the partitions are created.

Yugong provides replication service to execute the replication jobs. As shown in Figure 7, the replication service is implemented as a master-slave architecture. The RepMaster assigns projects evenly to RepSlaves and distributes the replication workloads to RepSlaves. RepSlaves obtain the partition replication information from the Plan Generator. Each RepSlave then subscribes to a Pub/Sub service in MaxCompute and will be notified whenever a partition belonging to some of its assigned projects is created. RepSlaves launch replication jobs for their assigned projects only, and they submit the jobs to MaxCompute's Global Job Scheduler. The Global Job Scheduler dispatches the jobs to the respective DCs, where the cluster scheduler (i.e., Fuxi [61]) allocates resources for the job execution.

A replication job copies a given table partition from the source DC to the destination DC. It is similar to a map-only MapReduce job with parallel replication instances. Before running the replication job, a planning job will first be launched to determine the number of replication instances, and then dispatch the files to these instances evenly. Each replication instance running in the source DC reads the files assigned to it locally and writes them to the remote destination DCs.

A RepSlave launches a replication job under one of the following three cases. The first one is *event-triggered*: when a partition that should be replicated is created, the RepSlave will be notified and launch the corresponding replication job. The second one is *on-demand*: when a job launching in the destination DC requires a partition and the partition is in the replication plan, but the partition is yet to be replicated, the RepSlave will be notified and launch a replication job to replicate the data. The third one is *scan*: the RepSlave also periodically scans the replication plan to check whether there are some partitions that need to be replicated (e.g., those delayed due to some scheduling issues). The on-demand replication jobs have the highest priority to be scheduled for execution.

6.3 Job Outsourcing Service

The job outsourcer has multiple instances, which obtain static job information from the MetaStore and dynamic cluster/network information from MaxCompute's Resource Monitor to make outsourcing decisions. Job outsourcers make their decisions independently for their own jobs according to the score function in Equation (18). Before outsourcing a job, a job outsourcer also first communicates with MaxCompute to check whether all partitions of tables required for the job are available in the destination DC.

If a job is to be outsourced, its job outsourcer sends all the necessary information for the job execution to the Global Job Scheduler, where the job will be submitted to the destination DC for execution. The outsourcer is also responsible for moving the output of the job back to the default DC of the job after its completion (note that this cost is small according to F2 and is included in $Cost(j, d)$ in Equation (18)).

Table 5: Statistics of DCs (DCs 3-7 reported as a range)

	#Project (100)	#Job (100K)	#Table (100K)	Normalized remote dependency
DC1	2	7	7	9
DC2	34	36	28	67
DCs3-7	1-4	3-18	5-23	1-14

7. PERFORMANCE RESULTS

We report some performance results [7] of Yugong in our geo-distributed DCs in this section. The results were obtained from 7 DCs, some of their statistics are given in Table 5. In Table 5, the *daily* remote dependencies among the DCs are normalized by the smallest daily amount of in-coming cross-DC bandwidth for a single DC. *The total amount of daily cross-DC dependencies is in the order of hundreds of PBs.*

While we report some results for all the 7 DCs in order to give a better overall picture, for other detailed results we only report for two DCs, DC1 and DC2, due to the page limit. DC1 is a medium-size DC hosting thousands of servers, while DC2 is a large DC with several tens of thousands of servers. DC2 has the largest numbers of projects, jobs, and tables, though this does not imply that DC2 is the largest DC in terms of computing resources.

7.1 Overall Performance

We first report the overall performance of Yugong running in production in Alibaba. Figure 8 shows the reduction of the in-coming cross-DC bandwidth usage for each DC on a typical day (i.e., the reduction by Yugong on that day is its typical performance). *Yugong reduced the cross-DC bandwidth usage from 14% to 88% for different DCs.* DC2 has the most significant reduction as it has the largest remote dependencies. *In total, 76% of the total bandwidth usage was reduced by Yugong on that day.*

Figure 9a plots the CDF of the completion time of the replication jobs in different DCs. The result suggests that most of the replication jobs can finish within a few minutes. *More than 96% of the jobs can finish in less than 600 seconds*, which is short compared with the running time of most analytics jobs in MaxCompute. Figure 9b reports the causes of the replication jobs. *About 62% to 82% of the jobs in different DCs are event-triggered by the creation of new time partitions. Few replication jobs are launched on-demand, meaning that few analytics jobs are delayed due to waiting for a required partition that is not yet replicated.*

[7] We remark that, wherever the performance results contain sensitive business/customer information, we report the relative numbers instead of the exact numbers. The relative numbers, however, are sufficient to show the improvements made by Yugong. *We highlight our main findings in italic fonts.*

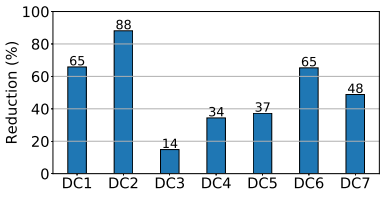
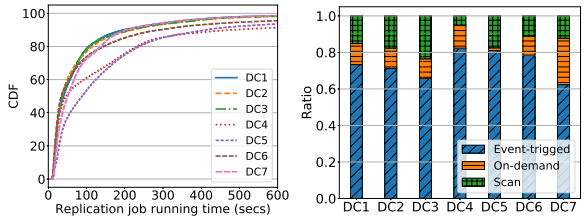


Figure 8: Cross-DC bandwidth reduction by Yugong



(a) Replication job time (b) Replication causes

Figure 9: Results of replication jobs (best viewed in color)

7.2 Performance of Table Replication

We then investigated how our table replication strategy performs under both fixed and dynamic table access patterns.

7.2.1 *Fixed Table Access Patterns*

We first assume that the table access patterns are fixed for different days. This scenario is used to test how well Algorithm 1 (denoted as **k-probe**) performs and how close it is to the optimal bandwidth usage under unlimited storage budget. We compared k-probe with a baseline counting algorithm (denoted as **Counting**) that ran in our production before the deployment of Yugong. Counting uses a simple counting and voting strategy to decide the life span: it evaluates the table access matrix $R_i^t(d)$ for each table i in the last 7 days and if a partition was read in 5 out of the 7 days, it is considered as worth-replicating. Then a life span for a table is formed if the worth-replicating partitions tend to be consecutive (i.e., 80% of the partitions in a time range are worth-replicating). Counting does not take the amount of dependencies into account and we cannot specify a storage budget for it. For fair comparison, we set the replication storage budget of k-probe to be the same as the storage used by Counting. We also report the optimal bandwidth cost one can achieve under unlimited storage budget, denoted as **Opt**.

Figure 10 shows the cross-DC bandwidth usage of Counting and k-probe in the 7 DCs, where the reported values are relative to the optimal bandwidth cost achieved by Opt. *The improvements of k-probe over Counting are from 18% to 45% for different DCs. In addition, k-probe's results are very close to the optimal ones.*

We then report how well k-probe performs under different replication storage budgets. We compared with another greedy method (denoted as **Greedy**), which greedily takes in a partition from the table access matrix that gives the current largest marginal gain until the storage budget is used up. Figure 11 gives the cross-DC bandwidth costs needed by each method under different replication storage sizes (relative to the actual storage size used by Opt). We report the cross-DC bandwidth costs as relative numbers to the total amount of cross-DC dependencies, meaning that the lower the relative cost, the more reduction in bandwidth usage is achieved. *Under the same storage size, k-probe uses 41% and 45% less bandwidth compared with Counting in DC1 and DC2, respectively. In addition, k-probe nearly achieves the optimal bandwidth usage given enough storage.* Without the life span concept, Greedy cannot further reduce the overall bandwidth usage even given more storage because it uses extra bandwidth to replicate old partitions from their default DCs every day.

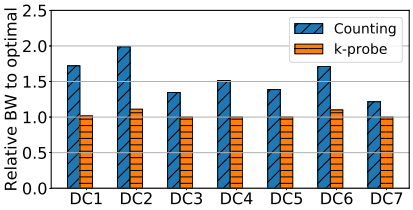
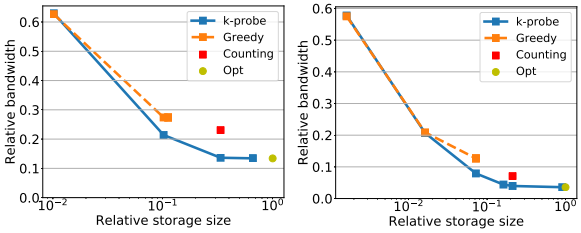
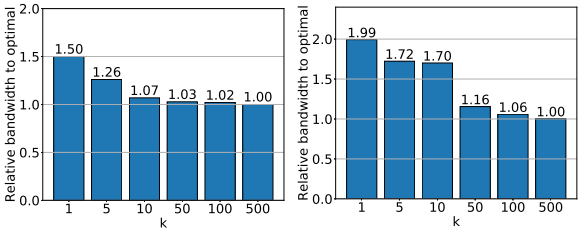


Figure 10: Cross-DC bandwidth usage with fixed access patterns (1.0 represents the optimal bandwidth usage)



(a) DC1 (b) DC2

Figure 11: Cross-DC bandwidth usage w/ different storage sizes



(a) DC1 (b) DC2

Figure 12: The effects of k on k-probe

Next we report the effects of the probing variable k on the performance of our k-probe algorithm. Figure 12 reports the cross-DC bandwidth usage (relative to the result of Opt) achieved by k-probe for different k, where the replication storage budget was set as the actual storage size used by Opt. The results vary for DC1 and DC2 for small values of k. For DC1, a small $k \approx 10$ already resulted in a good result, while for DC2 a small k resulted in relatively high cross-DC bandwidth usage as the storage budget was not fully used. This is because a small k may give a low marginal gain that stops the advance of the life span. *A relatively large k achieves performance close to the optimal bandwidth usage as stated in Theorem 1. We also note that the k-probe algorithm can finish in a few minutes even when we set $k = 500$.*

7.2.2 *Dynamic Table Access Patterns*

While in Section 7.2.1 we have assessed the effectiveness of our k-probe algorithm, showing that it can approach the optimal cross-DC bandwidth usage, the production environment is actually dynamic where the table access patterns are changing over time. We thus report the performance of our algorithm under the normal dynamic production environments in our DCs. This is also how we select the best strategy for production as the settings directly reflect the production scenarios. We used the table access patterns from production over 100 days and compared k-probe with the Counting

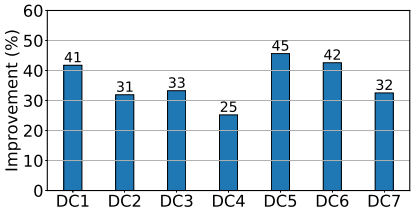
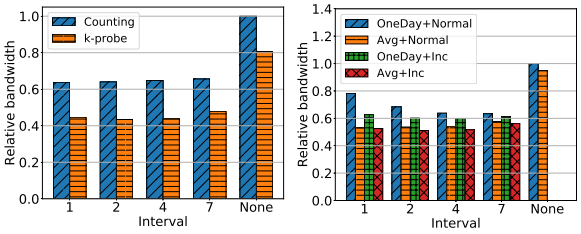


Figure 13: Cross-DC bandwidth usage reduction (over 100 days) of k-probe over Counting



(a) Varying update intervals (b) Different update strategies

Figure 14: Effects of update intervals and update strategies

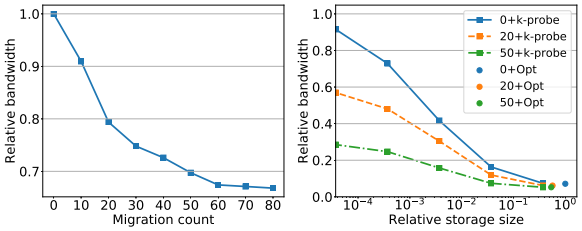
strategy previously used in our production. For fair comparison, we used the same amount of replication storage budget for both approaches. Since the dynamic production environments may require update maintenance of the replication plans, we also found the best update intervals for both approaches and used them in the performance evaluation.

Figure 13 reports the improvements achieved by k-probe over Counting on the reduction of cross-DC bandwidth usage in different DCs. *The results were obtained over 100 days and the improvements are from 25% to 45%. We remark that the amounts of reduced bandwidth are in the scale of several PBs to tens of PBs in different DCs per day.*

Next we report the effects of different update intervals on the performance. An update interval of 1 means updating daily and 7 means updating weekly, while *None* indicates that we never update the table replication plan. Figure 14a reports the total incoming cross-DC bandwidth usage of both approaches for different update intervals in DC2, where the numbers are relative to that of Counting with no update (i.e., None). The result shows that k-probe achieves considerably less total bandwidth usage over the 100 days compared with Counting using different update intervals. The improvements of k-probe over Counting is from 20% to 33%. *In general, setting the update interval to a small constant (e.g., 2 to 4) leads to better results for k-probe. Never updating the table replication plan performs the worst as it cannot adapt to the online workload changes.*

We also studied the performance of k-probe using different combinations of strategies for updating the table replication plan. We used two different table access matrices as the input to our algorithm: (1) $OneDay$ indicates that we used the table access matrix on that day as the input, and (2) Avg refers to using the average table access matrix over the last 7 days as the input. We used two different update strategies: (1) Inc updates the replication plan using the incremental strategy we proposed in Section 5.3, and (2) $Normal$ directly replaces the replication plan without considering the bandwidth cost for replicating the old partitions.

Figure 14b reports the total cross-DC bandwidth usage (over 100 days) using different strategies, where the numbers are relative to that of the $OneDay+Normal$ strategy with no update. The result



(a) Varying # of migrated projects (b) Different placement plans

Figure 15: Effects of varying number of migrated projects and different project placement plans (best viewed in color)

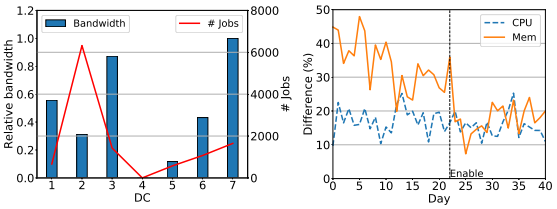
shows that using the $Avg + Inc$ strategy and updating the replication plan every two days achieve the best cross-DC bandwidth usage. *The Inc strategy performs better than $Normal$ because it considers the bandwidth cost for replicating the old partitions. Besides, Avg performs better than $OneDay$ as it can mitigate the oscillations in the table access patterns over different days.*

7.3 Performance of Project Migration

We first report the effects of the number of migrated projects on the cross-DC bandwidth usage given unlimited replication storage budget. Figure 15a shows that as we migrated more projects, the cross-DC bandwidth cost first decreased significantly, but then it only decreased moderately when we migrated more than 20 projects. Specifically, we reduced 21% of the bandwidth usage by migrating 20 projects and reduced 33% by migrating 60 projects. *The result suggests that our project migration strategy by migrating only a small number of projects can already significantly reduce the cross-DC bandwidth usage.*

We also examine the effectiveness of the decoupling of the original intractable problem (formulated in Section 5.1) into two simpler problems, i.e., first solve project migration under unlimited replication storage budget (Section 5.2) and then solve the table replication problem with the k-probe algorithm with a storage budget (Section 5.3). We generated three different project placement plans by migrating 0, 20 and 50 projects considering unlimited storage budget and only the top 1,000 tables with the most dependencies (accounting for 70% of the total cross-project dependencies). Then, we ran k-probe on each of the project placement plans with different storage budgets. We gave a total storage budget to all DCs instead of giving a storage budget to each DC individually to obtain an overview of the effects of project migration.

Figure 15b reports the cross-DC bandwidth usage using different project placement plans, which are relative to the total amount of cross-DC dependencies before any migration. We also plot 3 dots in the figure, i.e., 0+Opt, 20+Opt and 50+Opt, which represent the optimal cross-DC bandwidth usage given unlimited storage for each project placement. The result suggests that *under different project placement plans obtained by project migration, the optimal bandwidth usage given unlimited storage budget can be approached by our k-probe algorithm.* In addition, even with limited replication storage budgets, k-probe can always achieve less bandwidth usage using a better project placement plan (i.e., one that reduces more bandwidth usage) obtained under the assumption of unlimited storage budget. As from Figure 15a, we can see that migrating 50 projects is a better plan than migrating 20 projects, which is in turn better than no migration at all. Thus, Figure 15b shows that the results of 20+k-probe and 50+k-probe are always better than that of 0+k-probe. This also verifies the effectiveness of our decoupling strategy.



(a) Bandwidth usage reduction (b) Load balancing

Figure 16: Job outsourcing for cross-DC bandwidth usage reduction and load balancing

7.4 Performance of Job Outsourcing

We also evaluated the amount of cross-DC bandwidth that can be saved by scheduling jobs to run out of their default DCs. Figure 16a reports the number of jobs outsourced to each DC and the cross-DC bandwidth reduced by the outsourcing activities, where the numbers are relative to the maximum number of outsourced jobs to a DC and the maximum bandwidth reduction. There was no job that was outsourced to DC4 because other DCs had little dependency on DC4. *In total, job outsourcing can further reduce several hundred terabytes of bandwidth per day.*

We then report the benefit that job outsourcing can bring on load balancing among the DCs. Figure 16b depicts the differences between the CPU/memory utilization of two clusters over 40 days. The difference was calculated hourly between the utilization rates (0 to 100%) of CPU/memory in the two clusters, and the hourly differences were then averaged. During the period from Day 1 to Day 21, we can see that the difference in memory utilization rate is large (around 35 on average, meaning that if one cluster used 90% of its memory, the other one used only 55%). Indeed, what happened was that one of the clusters was memory-bound and the memory utilization often reached the limit. *We enabled job outsourcing on Day 22, and the difference of memory usage between the clusters decreased by about 2 times afterwards. This is because job outsourcing effectively outsourced some jobs from the heavily-loaded cluster and made better utilization of the memory of the other cluster.* The result demonstrates that job outsourcing can balance the resource utilization among different clusters in the DCs.

8. RELATED WORK

Geo-Distributed Scheduling. Recent work on geo-distributed scheduling for analytics workloads only consider a small number of jobs and assume that data are partitioned across geo-distributed DCs and tasks from a job can run in multiple DCs. Iridium [43] optimizes task scheduling and data placement to achieve low latency for analytic queries. Geode [56] and WANalytics [55] make query planning WAN-aware and provide network-centric optimizations for data analytics across DCs. Clarinet [54] proposes a WAN-aware query optimizer by considering the network bandwidth, task placement, network transfer scheduling and multiple concurrent queries. Tetrium [24] considers both the computing and network resources for task placement and job scheduling in geo-distributed DCs. Pixida [29] applies graph partitioning to minimize cross-DC task dependencies in data analytics jobs. Hung et al. [25] propose geo-distributed job scheduling algorithms to minimize the overall job running time but it does not consider the WAN bandwidth usage among the DCs. Bohr [34] exploits the similarity among data in different DCs for geo-distributed OLAP data cubes. Lube [57] detects and mitigates the bottlenecks in geo-distributed data analytic queries in real-time.

There are other works that make data stream analytics, distributed machine leaning and graph analytics WAN-aware. JetStream [44] proposes explicit programming models to reduce the bandwidth required for analyzing streaming datasets. For machine learning workloads, Gaia [23] and GDML [10] develop geo-distributed solutions to efficiently utilize the scarce WAN bandwidth while preserving the correctness guarantee of ML algorithms. Monarch [27] and ASAP [26] propose an approximate solution for geo-distributed graph pattern mining.

Cloud-Scale Data Warehousing. Google BigQuery [19], Amazon Redshift [4], Microsoft Azure Cosmos DB [39] and Alibaba MaxCompute [3] are large scale data warehouse products. The "project" concept in MaxCompute corresponds to the "database" in Redshift and the "project" in BigQuery. Even though Yugong is primarily built used as a plugin for MaxCompute in this work, similar ideas can also be applied to other geo-distributed data warehousing platforms.

Caching and Packing. Cache management is a well-studied topic across different levels of computer architecture, from CPU cache, memory cache, to application-level cache. Memcached [38] and Redis [46] are highly available distributed key-value stores providing in-memory cache over disks. EC-Cache [45] and SP-Cache [59] provide in-memory cache for data-intensive clusters and object stores. Piccolo [42], Spark [60], PACMan [6] and Tachyon [33] incorporate memory caching for cluster computing frameworks. In this work, we use disk storage as caching for remote partitions to reduce the cross-DC bandwidth. The table replication problem is a variant of the knapsack problem [7, 51] and the project placement problem is a variant of the bin packing problem [30, 51]. Tetris [21] analogizes the multiple resource allocation problem to the multi-dimensional bin packing problem for task scheduling.

Cluster Workload Analysis. There is a large body of work about cluster trace data analysis [47, 37, 5, 58, 14, 35, 1, 36]. We focus on analyzing the dependency relationship among jobs and tables in this work. The daily recurrent pattern of jobs was also found in [28].

9. CONCLUSIONS

We presented Yugong, which was developed to reduce the high and costly cross-DC bandwidth usage in Alibaba's geo-distributed DCs. Yugong works seamlessly with MaxCompute in very large scale production environments and has significantly reduced the cross-DC bandwidth usage by project migration (to reduce cross-DC dependencies), table replication (to eliminate heavy remote data reads) and job outsourcing (to move jobs to data and to balance resource utilization). We believe our solution will benefit both researchers and practitioners in work related to geo-distributed data placement and job scheduling.

In large scale production environments where there are many critical businesses such as those in Alibaba, system robustness is often more important than performance, and we have also observed that Yugong has consistently achieved stable performance over a long period of time (nearly 2 years). We are now also enhancing the connection between Yugong and our cluster scheduler, Fuxi, for very large scale geo-distributed job scheduling (e.g., tens of millions of jobs over tens of large clusters).

Acknowledgments. We thank the reviewers for their valuable comments. This work was supported by the National Key Research and Development Program of China (2016YFB1000503) and the Alibaba-CUHK collaboration project "System Optimizations and Next-Generation Computing Platforms" (code: 7010469).

10. REFERENCES

[1] Alibaba Cluster Trace. https://github.com/alibaba/clusterdata.

[2] Alibaba Datacenter Locations. https://www.alibabacloud.com/global-locations.

[3] Alibaba MaxCompute. https://www.alibabacloud.com/product/maxcompute.

[4] Amazon Redshift. https://aws.amazon.com/redshift.

[5] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben. On the diversity of cluster workloads and its impact on research results. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018.*, pages 533–546, 2018.

[6] G. Ananthanarayanan, A. Ghodsi, A. Warfield, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. Pacman: Coordinated memory caching for parallel jobs. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 267–280, 2012.

[7] R. Andonov, V. Poirriez, and S. V. Rajopadhye. Unbounded knapsack problem: Dynamic programming revisited. *European Journal of Operational Research*, 123(2):394–407, 2000.

[8] J. Baker, C. Bond, J. C. Corbett, J. J. Furman, A. Khorlin, J. Larson, J. Leon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, pages 223–234, 2011.

[9] Brazil to build undersea cables to Europe, Angola. https://phys.org/news/2014-01-brazil-undersea-cables-europe-angola.html.

[10] I. Cano, M. Weimer, D. Mahajan, C. Curino, G. M. Fumarola, and A. Krishnamurthy. Towards geo-distributed machine learning. *IEEE Data Eng. Bull.*, 40(4):41–59, 2017.

[11] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[12] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: yahoo!'s hosted data serving platform. *PVLDB*, 1(2):1277–1288, 2008.

[13] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. C. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 251–264, 2012.

[14] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 153–167, 2017.

[15] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 205–220, 2007.

[16] Decision Intelligence Lab, DAMO. https://damo.alibaba.com/labs/decision-intelligence.

[17] Five Google Subsea Cables Completed 2019. https://www.geo-tel.com/google-subsea-cables-completed-2019/.

[18] Global Internet Geography. https://www.telegeography.com/research-services/global-internet-geography/.

[19] Google BigQuery. https://cloud.google.com/bigquery.

[20] Google Datacenter Locations . https://www.google.com/about/datacenters/inside/locations/.

[21] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 455–466, 2014.

[22] A. G. Greenberg, J. R. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *Computer Communication Review*, 39(1):68–73, 2009.

[23] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu. Gaia: Geo-distributed machine learning approaching LAN speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 629–647, 2017.

[24] C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang. Wide-area analytics with multiple resources. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, pages 12:1–12:16, 2018.

[25] C. Hung, L. Golubchik, and M. Yu. Scheduling jobs across geo-distributed datacenters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing, SoCC 2015, Kohala Coast, Hawaii, USA, August 27-29, 2015*, pages 111–124, 2015.

[26] A. P. Iyer, Z. Liu, X. Jin, S. Venkataraman, V. Braverman, and I. Stoica. ASAP: fast, approximate graph pattern mining at scale. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018.*, pages 745–761, 2018.

[27] A. P. Iyer, A. Panda, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica. Monarch: Gaining command on geo-distributed graph analytics. In *10th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2018, Boston, MA, USA, July 9, 2018.*, 2018.

[28] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, I. Goiri, S. Krishnan, J. Kulkarni, and S. Rao. Morpheus: Towards automated slos for enterprise clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 117–134, 2016.

[29] K. Kloudas, R. Rodrigues, N. M. Preguiça, and M. Mamede. PIXIDA: optimizing data parallel jobs in wide-area data analytics. *PVLDB*, 9(2):72–83, 2015.

[30] R. E. Korf. A new algorithm for optimal bin packing. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.*, pages 731–736, 2002.

[31] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *Operating Systems Review*, 44(2):35–40, 2010.

[32] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with netstitcher. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, pages 74–85, 2011.

[33] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, November 3-5, 2014*, pages 6:1–6:15, 2014.

[34] H. Li, H. Xu, and S. Nutanong. Bohr: Similarity aware geo-distributed data analytics. In *9th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2017, Santa Clara, CA, USA, July 10-11, 2017.*, 2017.

[35] Q. Liu and Z. Yu. The elasticity and plasticity in semi-containerized co-locating cloud workload: a view from alibaba trace. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2018, Carlsbad, CA, USA, October 11-13, 2018*, pages 347–360, 2018.

[36] Z. Liu and S. Cho. Characterizing machines and workloads on a google cluster. In *41st International Conference on Parallel Processing Workshops, ICPPW 2012, Pittsburgh, PA, USA, September 10-13, 2012*, pages 397–403, 2012.

[37] C. Lu, K. Ye, G. Xu, C. Xu, and T. Bai. Imbalance in the cloud: An analysis on alibaba cluster trace. In *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*, pages 2884–2892, 2017.

[38] Memcached. `https://memcached.org/`.

[39] Microsoft Azure Cosmos DB. `https://azure.microsoft.com/services/cosmos-db/`.

[40] Microsoft Datacenter Locations . `https://azure.microsoft.com/en-us/global-infrastructure/regions/`.

[41] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling memcache at facebook. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*, pages 385–398, 2013.

[42] R. Power and J. Li. Piccolo: Building fast, distributed programs with partitioned tables. In *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, pages 293–306, 2010.

[43] Q. Pu, G. Ananthanarayanan, P. Bodík, S. Kandula, A. Akella, P. Bahl, and I. Stoica. Low latency geo-distributed data analytics. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, pages 421–434, 2015.

[44] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman. Aggregation and degradation in jetstream: Streaming analytics in the wide area. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, pages 275–288, 2014.

[45] K. V. Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran. Ec-cache: Load-balanced, low-latency cluster caching with online erasure coding. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 401–417, 2016.

[46] Redis. `https://redis.io/`.

[47] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012*, page 7, 2012.

[48] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013*, pages 351–364, 2013.

[49] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband internet performance: a view from the gateway. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, pages 134–145, 2011.

[50] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive - A warehousing solution over a map-reduce framework. *PVLDB*, 2(2):1626–1629, 2009.

[51] P. H. Vance. Knapsack problems: Algorithms and computer implementations (S. martello and p. toth). *SIAM Review*, 35(4):684–685, 1993.

[52] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop YARN: yet another resource negotiator. In *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pages 5:1–5:16, 2013.

[53] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*, pages 18:1–18:17, 2015.

[54] R. Viswanathan, G. Ananthanarayanan, and A. Akella. CLARINET: wan-aware optimization for analytics queries. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 435–450, 2016.

[55] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese. Wanalytics: Analytics for a geo-distributed data-intensive world. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[56] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese. Global analytics in the face of bandwidth and regulatory constraints. In *12th USENIX Symposium on*

Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015, pages 323–336, 2015.

[57] H. Wang and B. Li. Lube: Mitigating bottlenecks in wide area data analytics. In *9th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2017, Santa Clara, CA, USA, July 10-11, 2017.*, 2017.

[58] J. Wilkes. More Google cluster data. Google research blog, Nov. 2011. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`.

[59] Y. Yu, R. Huang, W. Wang, J. Zhang, and K. B. Letaief. Sp-cache: load-balanced, redundancy-free cluster caching with selective partition. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*, pages 1:1–1:13, 2018.

[60] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 15–28, 2012.

[61] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale. *PVLDB*, 7(13):1393–1404, 2014.