Wide-Area Analytics with Multiple Resources

Chien-Chun Hung
USC
chienchun.hung@usc.edu

Ganesh Ananthanarayanan
Microsoft
ga@microsoft.com

Leana Golubchik
USC
leana@usc.edu

Minlan Yu
Harvard
minlanyu@seas.harvard.edu

Mingyang Zhang
USC
mingyangz@usc.edu

Abstract

Running data-parallel jobs across geo-distributed sites has emerged as a promising direction due to the growing need for geo-distributed cluster deployment. A key difference between geo-distributed and intra-cluster jobs is the heterogeneous (and often constrained) nature of compute and network resources across the sites. We propose Tetrium, a system for multi-resource allocation in geo-distributed clusters, that jointly considers both compute and network resources for task placement and job scheduling. Tetrium significantly reduces job response time, while incorporating several other performance goals with simple control knobs. Our EC2 deployment and trace-driven simulations suggest that Tetrium improves the average job response time by up to 78% compared to existing data-locality-based solutions, and up to 55% compared to Iridium, the recently proposed geo-distributed analytics system.

1 Introduction

Large online service providers like Microsoft, Google, Amazon and Facebook are deploying tens of datacenters and many hundreds of smaller "edge" clusters globally to provide their users with low latency access to their services [2, 7, 35]. These *geo-distributed sites* continuously generate data both about the services deployed on them (like end-user session logs) as well as server health (like performance monitor logs). Collectively analyzing this geo-distributed data is crucial for many operational and management tasks. Examples of such tasks include analyzing server logs to maintain system health dashboards, analyzing user logs to make advertisement choices, and picking relay servers for online services using network performance logs [37, 42].

As the results produced by these *analytics queries* are used for making critical decisions by data analysts and real-time applications, minimizing their response times is the key. Recent efforts in geo-distributed analytics have demonstrated that centrally aggregating all the data to a single site and then analyzing them can seriously limit the timeliness of the analytics for the above applications. In addition, they lead to wasteful use of the WAN bandwidth [17, 47]. It has emerged that executing the queries *geo-distributed* by leaving the data in-place at the sites can lead to faster query completions [17, 32, 47, 55].

An important characteristic of geo-distributed clusters is *heterogeneity: in compute as well as network resources*. The bandwidth capacities of the sites may vary by an order of magnitude [18, 47, 54]. Compute capacities are also highly diverse. Conversations with one of the largest online service providers (OSP) [10] reveal that the heterogeneity in compute capacity (cores and memory) among its datacenters varies by *two orders of magnitude*. Further, the availability of network and compute resources also varies depending on utilizations. An additional source of heterogeneity is the non-uniform distribution of a job's input data across sites; e.g., when analyzing user session logs of Bing within the last two hours, more user data is likely to be present on sites where it is working hours for their nearby users than night times. (See §2.1 for details.)

Recent works on geo-distributed data analytics only recognized the network heterogeneity. These solutions minimize network transfers alone by placing mapper tasks at the sites where their input is located and placing reducer tasks to minimize shuffle time [17, 39, 47, 54, 55]. Their design assumes that the sites are indistinguishable in their compute capacities and have effectively infinite capacities to run the tasks. As described above, the former is invalid in real deployments [10]. Further,

EuroSys '18, April 23–26, 2018, Porto, Portugal

ACM ISBN 978-1-4503-5584-1/18/04…$15.00
https://doi.org/10.1145/3190508.3190528

prior work on data analytics, even within single DCs, have documented "multi-waved" execution of analytics jobs of even a single stage (e.g., reduce stage) due to the constraints on compute capacities (i.e., only a fraction of the job's tasks execute simultaneously per "wave") [14, 15, 45].

Placing tasks only for minimizing network transfer delay can lead to more tasks being scheduled at a site than available compute slots, which causes multiple waves of execution *even when* slots are available at other sites (§2.2). Not only does this inflate job response time, it also introduces *inaccuracy in its model of network bandwidths* when the reduce tasks of subsequent waves execute. This is because their network traffic was not accounted to occur at the time of the later waves. This error in bandwidths cascades and degrades further scheduling decisions.

In this paper, we take the first effort towards *allocating multiple resources – compute slots and network – to data analytics jobs with parallel tasks across heterogeneous geo-distributed clusters*. The problem of multi-resource allocation is relatively easier in intra-DC analytics due to the near homogeneity between machines (or racks) in compute and network capacities.

While at first blush, the problem appears to be that of multi-dimensional bin packing of balls (tasks), there are key differences which make theoretical packing heuristics hard to adapt. The resource demands of tasks in our setup are not static but depend on the site (bin) where the task is placed. For instance, the WAN usage of a map or reduce task depends on the amount of its input data present on its own local site. Even recent systemic solutions for intra-DC packing [28–30], fall short for a geo-distributed setting because they model *remote network accesses* of tasks (e.g., reading data from a remote site) with a simple and fixed "penalty". While such a fixed penalty works for a homogeneous setting, this is a crucial omission in the heterogeneous geo-distributed setup.

Further, the heterogeneity among the geo-distributed sites means that we have to jointly make decisions on all the tasks of a job so that all of them face equal resource contention and finish together. The duration of a job with many parallel tasks depends on the last task's completion. Note that this scheduling requirement is orthogonal to straggler mitigation via speculation [15, 16, 57] and data skew addressal [40, 41]. Finally, it is also important to optimize the scheduling *across* jobs since they compete for the same set of slots and bandwidth. In doing so, we optimize for the same metrics of interest to intra-DC analytics – response time and fairness.

Unfortunately, the problem of allocating multiple resources of geo-distributed sites for simultaneous jobs of many parallel tasks is computationally intractable. Hence, we devise an efficient heuristic.[1] We formulate the multi-resource scheduling problem for optimizing a single job's response time, by placing its tasks across the sites, as a linear program (LP). An important aspect of our formulation is the modeling of multiple waves of execution among tasks within a stage, and hence also decide the *ordering* in which the tasks are scheduled in each stage. We have different formulations for the map stages (one-to-one mapping) and reduce stages (many-to-many shuffling), based on their communication patterns across the sites. We believe this is the first multi-wave modeling proposed for multi-resource scheduling.

To schedule multiple geo-distributed jobs, we integrate the above LP with the Shortest Remaining Processing Time (SRPT) heuristic. The LP described helps us accurately and efficiently identify jobs with the least remaining time, instead of proxies like remaining number of tasks. It also re-evaluates its task placement based on resources consumed by tasks of other simultaneous jobs in each scheduling instance. In doing so, it balances the goals of response time, makespan and fairness.

Our solution is careful about the usage of WAN bandwidth, i.e., bytes transferred across sites. As WAN bandwidth is the important resource especially in wide-area analytics [17, 55], our solution incorporates a *WAN usage budget* for the scheduling decisions. A simple knob in our solution trades off between optimizing job response times and WAN usage.

We have built Tetrium, a system for multi-resource allocation in geo-distributed clusters inside Apache Spark [5]. We evaluate Tetrium using (a) an Amazon EC2 deployment across several geographically distributed regions [2] running the TPC-DS [8] and the Big Data benchmarks [3], as well as (b) large-scale simulation experiments driven by production traces. Tetrium improves the average job response time by up to 78% compared to existing locality techniques [5, 56], up to 55% compared to Iridium [47], and 33% compared to Tetris [28].

2 Motivation

We first motivate the heterogeneity in both geo-distributed clusters for analytics jobs (§2.1). Next, we use an example to show the key challenges in scheduling jobs for such heterogeneous settings (§2.2).

2.1 Geo-distributed data analytics

Architecture Figure 1 shows the architecture of a geo-distributed analytics framework that logically spans multiple sites (datacenters of different sizes). Each site contains multiple compute slots (corresponding to some amount of memory and cores), and diverse uplink and

[1] We leave the discussion about optimality in Section §8.

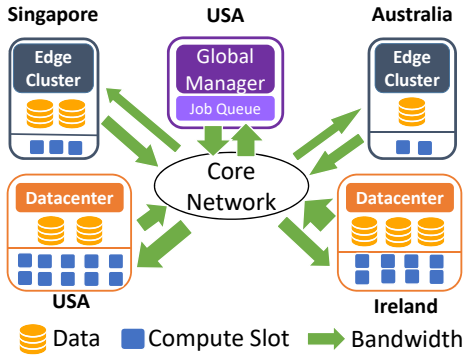
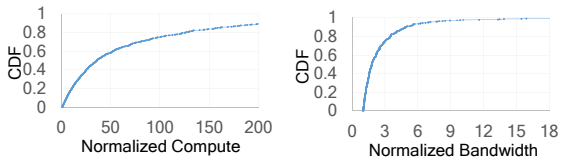


Figure 1: Wide-area data analytics across geo-distributed cluster. Analytics jobs are submitted to the global manager, and may require data stored across geo-distributed sites which have various capacities in compute slots and bandwidth.



(a) Compute (b) Network

Figure 2: Heterogeneity in compute and network capacities among hundreds of sites of one of the largest online service providers (OSP). Both graphs are normalized to the corresponding lowest values.

downlink bandwidths. We assume all sites are connected with a congestion-free network in the core like prior work [47], which is validated by measurement studies [4]. Data can be generated at any site and the input data required by a job may be located across different sites. A centralized *global manager* accepts analytics jobs and translates them into a DAG of stages with multiple parallel tasks. Minimizing the response times is the key for geo-distributed analytical jobs because the output of these jobs are often used for making critical and real-time decisions.

Heterogeneity across Clusters Aggregating all the data to a central location is wasteful since we do not know beforehand the datasets that get accessed; most data (82%, as per our analysis, in a large OSP's big data cluster) is never accessed or accessed very few number of times (Figure 1 in [12] says that 80% of data is accessed ≤ 2 times over a five day period in Microsoft Cosmos). Prior work has demonstrated that aggregating data *after the query arrives* seriously limits the timeliness of the analytics [17, 47]. The better approach to geo-distributed data analytics is leaving the data "in place" and distribute the tasks of a job across the different clusters. Yet, the key challenge in this approach is *heterogeneity* across the clusters, as we elaborate on next.

(1) Heterogeneous data sizes: Data generated at the sites also vary. For a globally deployed service across many sites, the size of the user session logs at the sites is dependent on the number of sessions served by the site, which naturally have significant variation. Our analysis of Skype logs on call performance generated at over 100 different Azure sites (relays) shows considerable variation: relative to the site with the minimum data generated, the median, 90th percentile and maximum values are 8×, 15× and 22× more [37]. These logs are constantly analyzed for dashboards and system health.

It is difficult to provision the sites with compute capacity proportional to the data generated for the following two reasons. First, the distribution of data sizes vary over time and is not a constant. Second, the distribution of data across sites *for a given job* might be vastly different than the overall distribution of data size. As a result, a job may have imbalance in computing slots compared to the computing capacity at sites.

(2) Heterogeneous compute capacities: Figure 2(a) shows that the compute capacities of clusters of one of the largest online service providers (OSP). We see that compute capacities differ by up to two orders of magnitude across hundreds of sites. This is because clusters are built at different times with different levels of investment, different capacity requirements, are constrained by site size, energy, cooling, etc. The impending trend of edge computing [1, 9] will only increase the heterogeneity since edge clusters are envisioned to be just a few racks of servers, as opposed to large data centers.

The *available* capacities tend to vary even more [18, 47]: heavy usage of one or a few clusters by *non-analytics* jobs leads to limited availability in them. Clusters are often not provisioned exclusively for analytics but share resources, e.g., with client-facing services. When the client load to these services increase, the resources available for data analytics shrinks and contributes to the heterogeneity.

(3) Heterogeneous network bandwidths: Figure 2(b) measures the skew in network bandwidth across different inter-site links of the large OSP. Note the 18× variation in bandwidths normalized to the least value. In fact, recent work has shown similar heterogeneity across Amazon EC2 sites of over 25× in their bandwidths [54]. Equal spreading of tasks across all sites, as is recommended by prior approaches [16, 53], will bottleneck sites with lower bandwidths. Therefore, we should also consider network resources for geo-distributed jobs.

Notation	Explanation
S_x	number of compute slots at site x
B_x^{up}, B_x^{down}	uplink/downlink bandwidth at site x
I_x^{input}, I_x^{shufl}	input/intermediate data at site x
M_x, R_x	number of map-/reduce-tasks placed at site x
t_{map}, t_{red}	computation time of a map-/reduce-task
T_{aggr}	network transfer duration of map-stage
T_{map}	computation duration of map-stage
T_{shufl}	network transfer duration of reduce-stage
T_{red}	computation duration of reduce-stage

Table 1: Definition of Notations.

To sum up, the heterogeneity mentioned above makes it difficult to build geo-distributed clusters that match available resources to data distribution. Addressing such heterogeneity in resource scheduling decisions is the focus of this work.

2.2 Illustrative Examples

We now motivate the needs for jointly scheduling compute and network resources through illustrative examples, and show how state-of-the-art solutions fall short. We consider a geo-distributed setup that has varying number of compute slots (S_x) at each site x as well as uplink/downlink bandwidth capacities (B_x^{up} and B_x^{down}); the volume of data I_x stored at each site might be unevenly distributed. Table 1 summarizes our notation.

Joint Compute- & Network-Aware Task Placement: Figure 4 specifies a 3-site example setup, in which site-1 is the most powerful in terms of number of compute slots and bandwidth, while it has the least amount of input data compared to the other 2 sites. We consider an analytical job with one map and one reduce stage at the ease of explanation, while the insights can be generalized to jobs with multiple stages. Each map task processes 100 MB input data and takes 2 sec to finish; after map stage, the intermediate data only has half of the size of input data. There are 500 tasks in reduce stage, and each reduce task takes 1 sec to finish. The state-of-the-art solution of assigning tasks across the sites is *Iridium* [47], which processes all the map tasks locally and then decide the best placement of reduce tasks that minimizes network resources. In the map stage, the computation bottleneck is at site 2: $2ses \cdot \lceil \frac{300}{10} \rceil = 60$ sec. In reduce stage, Iridium places the reduce tasks so that the shuffle time is minimized. The shuffle bottleneck in reduce stage is at site 2, in which its downloading takes $\frac{10+25GB \cdot 0.3}{1GBps} = 10.5$ sec, and its uploading also takes $10.5sec$ without going into details. The computation bottleneck in reduce stage is at site 3: 1 sec $\cdot \lceil \frac{350}{20} \rceil = 18$ sec. The end-to-end job completion time for this job is therefore 88.5 sec.[2]

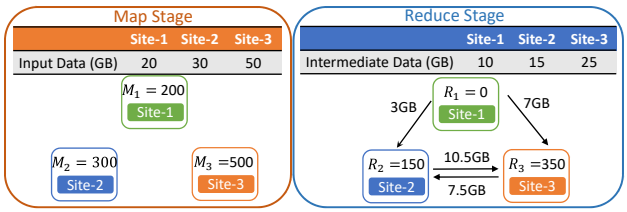
A better placement in this example considering both network and computation capacity would transfer some input data from site-2 (15.7 GB) and site-3 (21.4 GB) to site-1 as site-1 has powerful computation capacity. Despite the minor increment (15.7 sec) in transferring input data to site-1, the better assignment balances the computation workloads among the sites so that the computation duration in map-stage drops from 60 sec to 30 sec. In reduce-stage, it again reduces the computation duration from 18 sec to 8 sec by considering both network and computation capacity. Note that although Iridium optimizes shuffle time, the better assignment in this example has a better intermediate data distribution which leads to faster shuffle time (6.13 sec) compared to Iridium (10.5 sec). The end-to-end job completion time for this placement is 59.83 sec, which is only 68% of Iridium's completion time. Figure 3 specifies the placement and numbers in details.

There are two key insights by comparing the two solutions. First, in the map phase, it is sometimes beneficial when we shift some workload away from the compute bottlenecked site (i.e., site 2) to other sites for a small increase in network transfer. Note that the opposite solution of aggregating all input data to the most powerful site, referred as *Central* approach, is far from optimal as well. Central approach would aggregate all input data to site-1, and runs all the computation there without having to transfer data across the sites again. The end-to-end job completion time based on Central approach, without going into details, is 93 sec, which is $1.55\times$ of the better assignment mentioned above. Second, in the reduce stage, although Iridium aims at minimizes the network transfer time by avoiding data transfer for map stage and optimizes the shuffle time for reduce stage, it does not consider the computation capacity at each site (especially site 3). This example highlights the importance of coordinating the heterogeneous capacity across the sites, as well as jointly considering network and compute resources to optimize for job completion time.

Joint Job Scheduling & Task Placement:

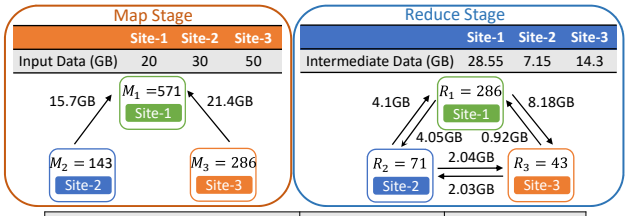
Task placement under multiple resource constraints becomes more intricate when multiple simultaneous jobs compete for resources. Placing the tasks of the jobs according to each job's optimal placement may no longer be the best option. We explain this using an example

[2]Here we are calculating the total duration based on worst-case estimation that there is no overlap between the network transfer time and computation time in each stage. In practice, a task starts computation, once its data are gathered, without waiting for the network transfer time of the other tasks.



	Map Stage			Reduce Stage		
	Site-1	Site-2	Site-3	Site-1	Site-2	Site-3
Network Transfer Time (sec)	0	0	0	2	10.5	3.75
Computation Time (sec)	10	60	50	0	15	18
Combined Duration (sec)	0 + 60 + 10.5 + 18 = 88.5					

(a) Iridium



	Map Stage			Reduce Stage		
	Site-1	Site-2	Site-3	Site-1	Site-2	Site-3
Network Transfer Time (sec)	7.42	15.7	10.7	2.45	6.13	5.11
Computation Time (sec)	30	30	30	8	8	8
Combined Duration (sec)	15.7 + 30 + 6.13 + 8 = 59.83					

(b) Better Approach

Figure 3: Task Placement Results: Iridium (left) and A Better Approach (right). M_i and R_i are the numbers of map and reduce tasks at site i, respectively. The results of network transfer and computation time, as well as the total time, are listed in the table below the figures. The better approach finishes the job faster by substantially reducing the bottlenecked computation time (e.g., site-2 in map stage) while incurring marginal network delay.

	Site-1	Site-2	Site-3
Number of Compute Slots, S_x	40	10	20
Uplink Bandwidth (GB/s), B_x^{up}	5	1	2
Downlink Bandwidth (GB/s), B_x^{down}	5	1	5
Input Data (GB), I_x^{input}	20	30	50

Figure 4: Bandwidth, compute capacities and input data for our three-site example setup.

with 3 sites and each has 3 compute slots and 1 GBps upload/download bandwidth. There are two jobs (job-1 and job-2) in this example, each with 3 and 12 tasks respectively. For simplicity, both jobs contain only a map-stage, and each task runs for 1 sec. Job-1's input data require 0, 1 and 2 tasks across the 3 sites, while job-2 requires 2, 4, and 6 tasks. The optimal placement for each job *in isolation* would run all the tasks locally without any data transfer, i.e., $M_1, M_2, M_3 = 0, 1, 2$ for job-1 and $2, 4, 6$ for job-2, which leads to response time 1 sec for job-1 and 2 sec for job-2 (since job-2 requires 2 waves of computation). When the 2 jobs are jointly scheduled, their best placement would depend on the order of the jobs. Scheduling job-1's tasks prior to job-2 will not change job-1's task placement, but the best placement for job-2 becomes $6, 4, 2$ with response time $2.4sec$[3]; the average duration for the two jobs is 1.7 sec. The opposite ordering - job-2's tasks followed by job-1's tasks - will lead to optimal placement for job-2, while the best placement for job-1 becomes $3, 0, 0$ with response time 3.3 sec; this leads to worse average duration 2.65 sec.

Two takeaways from this example: (1) The optimal schedule (in this example) was obtained *without either of the jobs* achieving their individual optimal task placement due to the inter-job contention for resources. (2) Furthermore, an optimal scheduling order of the jobs is a complex interaction between the available slots, available network bandwidths, and data distribution.

[3]The bottleneck for job-2 is at site-1: transferring data for 4 remote tasks takes $4 \cdot 100$ MB 1 GBps = 0.4 sec while finishing 6 tasks takes $2sec$, i.e., 2 waves; it adds up to 2.4 sec in response time.

3 Compute/Network-Aware Task Placement

Our scheduling decisions happen upon the arrivals of new jobs, or when occupied resources are released, e.g., completion of jobs. We first describe our compute- and network-aware task placement, then integrate it with job scheduling in §4.

Task placement decisions essentially determine the following: (1) at which site should a task be placed, and (2) from which site should a task read data. Moreover, tasks in a stage often run across multiple waves as the compute slots are insufficient for launching all tasks at once. Therefore, the decisions should also include the ordering of the tasks. Considering task placement and ordering together is, however, challenging even for a single stage. The formulation is a mixed integer linear program with Omn variables where m is the number of tasks and n is the number of sites, and is inefficient to solve in the time scales (of seconds) required for cluster schedulers.

Our approach is to solve task placement first based on a linear program under heterogeneous resource constraints, for each stage *independently* – map-stage (§3.1) and reduce-stage (§3.2). We then address task ordering within each stage in §3.3, and address sub-optimalities due to solving each stage independently (§3.4).

3.1 Map-Task Placement

In placing the map-tasks, we can view our problem as determining what fraction of the job's tasks ($m_{x,y}$) should run at site y with corresponding data residing at site x.

$_{y\neq x} m_{x,y}$ denotes the fraction of tasks that are not placed at site x but need to read data from site x. The amount of data to be transferred out of site x is then $I^{input} \cdot {}_{y\neq x} m_{x,y}$, where $I^{input} = {}_x I_x^{input}$ is the total volume of input data. Therefore, the upload transfer time at site x is $\frac{I^{input} \cdot {}_{y\neq x} m_{x,y}}{B_x^{up}}$ given site x's upload bandwidth B_x^{up}. Similarly, the fraction of map-tasks that are placed at site x but need to read data from the other sites $y \neq x$ is $_{y\neq x} m_{y,x}$, so the download transfer time at site x is $\frac{I^{input} \cdot {}_{y\neq x} m_{y,x}}{B_x^{down}}$.

The number of map-tasks at site x can be denoted by $n_{map} \cdot {}_y m_{y,x}$, assuming n_{map} is the total number of map-tasks. Given that site x has S_x slots, it takes $\frac{n_{map} \cdot {}_y m_{y,x}}{S_x}$ waves to finish all the map-tasks at site x. Hence, the computation time at site x is $t_{map} \cdot \frac{n_{map} \cdot {}_y m_{y,x}}{S_x}$, assuming each map-task's duration is t_{map} at the ease of presentation; we deal with variances in task durations in §5.

Based on the principled guidelines provided above, we can then formulate the map-task placement problem into the following Linear Program (LP) to minimize the job's remaining processing time at map-stage.

LP: map-task placement

$$min \quad T_{aggr} + T_{map} \tag{1}$$

$$s.t. \quad T_{aggr} \geq \frac{I^{input} \cdot {}_{y\neq x} m_{x,y}}{B_x^{up}}, \forall x \tag{2}$$

$$T_{aggr} \geq \frac{I^{input} \cdot {}_{y\neq x} m_{y,x}}{B_x^{down}}, \forall x \tag{3}$$

$$T_{map} \geq t_{map} \cdot \frac{n_{map} \cdot {}_y m_{y,x}}{S_x}, \forall x \tag{4}$$

$$m_{x,y} \geq 0, {}_y m_{x,y} = \frac{I_x^{input}}{I^{input}}, {}_{x\,y} m_{x,y} = 1, \forall x, y \tag{5}$$

Here, our goal (Eq. 1) is to minimize the map-stage's total processing time which consists of both the time it takes to move input data across sites (T_{aggr}) and the computation time of all map tasks (T_{map}).[4] The constraints in Eqs. 2 and 3 reflect the aggregation time, i.e., time to transfer the input data to where the map tasks are placed. Since the network transfer time is dominated by the bottleneck site, T_{aggr} is at least as large as the upload and download duration at each site. Eq. 4 reflects that the map-stage's computation time T_{map} is dominated by the maximum computation time across all the sites.

Note that, to obtain an LP formulation above (rather than a MILP), we focus on the *fraction* (rather than the number) of tasks to be run at each site x. Of course, the number of tasks at each site x needs to be integral; hence, we round the solution. With a sufficiently large number of tasks per job, this approximation should not significantly affect performance.

Solving the above LP gives us the following: (a) The minimum remaining processing time ($T = T_{aggr} + T_{map}$) for the map-stage. (b) The fraction of map-tasks to place at each site y that needs input data from x ($m_{x,y}$); in essence, this provides the list of tasks at each site. (c) The needed slot allocation ($D = \{d_x = \min S_x, {}_y m_{y,x} \cdot n_{map}, \forall x\}$), which is used in job scheduling decisions (§4) Note that this formulation is deterministic in nature, i.e., it relies on averages and as such does not reflect the variance in task duration that may occur due to data skew and availability of network capacity (when reading from remote sites). We discuss how to handle the variance in §5.

[4] In data analytics frameworks, the aggregation and map stages do not overlap because it is hard to track when the map data is ready.

3.2 Reduce-Task Placement

Different from the map-stage, each reduce-task reads data from the output of all map-tasks. Hence, in placing reduce-tasks, we only need to identify the fraction of reduce-tasks r_x to be placed at each site x.

Since site x has r_x fraction of the reduce-tasks, it needs to process r_x fraction of total intermediate data for reduce-stage: the volume of data to be transferred out of site x is $I_x^{shufl} \cdot 1 - r_x$, and the volume of data to be transferred to site x is $_{y\neq x} I_y^{shufl} \cdot r_x$. Therefore the upload and download duration at site x are $\frac{I_x^{shufl} \cdot 1 - r_x}{B_x^{up}}$ and $\frac{{}_{y\neq x} I_y^{shufl} \cdot r_x}{B_x^{down}}$, respectively. The number of reduce-tasks at site x is $n_{red} \cdot r_x$ assuming n_{red} is the number of reduce-tasks, and it takes $\frac{n_{red} \cdot r_x}{S_x}$ waves to finish all the reduce-tasks. Therefore, the computation time at site x is $t_{red} \cdot \frac{n_{red} \cdot r_x}{S_x}$; similar in §3.1, we assume constant task duration t_{red}.

LP: reduce-task placement

$$min \quad T_{shufl} + T_{red} \tag{6}$$

$$s.t. \quad T_{shufl} \geq \frac{I_x^{shufl} \cdot 1 - r_x}{B_x^{up}}, \forall x \tag{7}$$

$$T_{shufl} \geq \frac{{}_{y\neq x} I_y^{shufl} \cdot r_x}{B_x^{down}}, \forall x \tag{8}$$

$$T_{red} \geq t_{red} \cdot \frac{n_{red} \cdot r_x}{S_x}, \forall x \tag{9}$$

$$r_x \geq 0, {}_x r_x = 1, \forall x \tag{10}$$

Our goal (Eq. 6) is to minimize job's remaining processing time in the reduce-stage, i.e., the sum of the network shuffle time (T_{shufl}) and reduce computation time (T_{red}).

Eq. 7 and Eq. 8 bound the shuffle time T_{shufl} by the network transfer duration at each site: T_{shufl} is dominated by the maximum of upload and download duration across all the sites. Eq. 9 reflects that the computation time of the reduce-stage is dominated by the maximum computation time across all the sites. Note that our formulation for the reduce stage is similar to the model proposed in [47]. The key difference is that we extend the model to jointly minimize the time spent in network transfer and in computation time.

The outcome of solving this LP gives us the following: (a) The optimized remaining processing time ($T = T_{shufl} + T_{red}$) for the reduce-stage. (b) The fraction of

reduce-tasks to place at each site x (r_x). (c) The needed slot allocation ($D = \{d_x = \min S_x, r_x \cdot n_{red}, \forall x\}$).

As in the case of map tasks, the LP formulation produces a fraction of reduce tasks to place at each site, which (using similar rationale) is rounded to obtain an integral number of reduce tasks to place at each site x.

Our task placement model extends the model proposed in Iridium [47]. Iridium assumes there are sufficient compute slots and all tasks can start at once without queuing delay; hence Iridium focuses on minimizing network delay only. Our model targets on the general setting in which both computation and network transfer contribute the overall delay: hence our model focus on minimizing the sum of network and computation delay, where the computation delay is estimated on multi-wave model. In addition, we propose a task placement model for map stage, which has different data transfer pattern and is not addressed by Iridium. We compare our solution with Iridium in Evaluation Section (§6).

3.3 Task Ordering

When the compute slots are constrained, it may take several waves to finish all a job's tasks. Therefore, selecting the set of tasks to run in each scheduling instance is critical for completing their associated jobs quickly.

With tasks of varying durations, the key principle for minimizing the associated job's response time is to start the tasks with long duration first to avoid having long-duration tasks delay job response time [38]. For map-stages, since the tasks that read data from remote sites take significantly longer to complete compared to the tasks that run locally with the data, we start the remote tasks first before the local tasks. Specifically, we first select the tasks that takes longest time to fetch its input data, i.e., the tasks that read data from the site with the most constrained upload bandwidth, as each map-task generally processes the same size input partition and the only factor in different input fetching time is the bandwidth between the two sites. We further reduce the network contention by spreading the remote tasks launching across different sites, as opposed to launching all the remote tasks reading data from the most constrained site at once.

For reduce-stages, we also start with the longest-duration tasks based on their network transfer time. Note that the LP formulation in Section §3.2 assumes each reduce-task processes the same volume of intermediate data; in practice, each reduce task may have different amount of data because the intermediate data may not be equally partitioned across the keys. Therefore, we order the tasks based on their size of input data: the larger the task's input data size, the earlier it gets launched.

We verify in §6 that our task ordering design does reduces a job's response time; we further describe how we adapt this design to deal with dynamic slot arrivals in §5.

3.4 Mismatch between Map and Reduce

The intermediate data distribution depends on task placement decisions made for the map-stage. Tetrium's stage-by-stage approach falls short of addressing such a dependency as we place map-tasks by optimizing map-stage's duration ($T_{aggr} + T_{map}$), without considering the consequences for the reduce-tasks. This could (potentially) result in a longer reduce-stage ($T_{shufl} + T_{red}$). A better approach could be to decide the placement of the tasks of map and reduce stages *jointly*. Without specifying the details, such alternative leads to job duration of 44.875 in our previous example in §2, as opposed to 50.88 by Tetrium.

Scheduling a job's end-to-end tasks based on its full DAG description has been an open (and hard) problem in the literature, with previous efforts often resorting to a stage-by-stage approach [47, 48]. Here we investigate to what extent Tetrium's particular stage-by-stage approach is handicapped by not considering a full DAG-based formulation. We use an *unrealistic* alternative by assuming having full information about all tasks (map and reduce) upfront, and design a heuristic that produces a more favorable (to the reduce-stage) intermediate data distribution. Instead of Tetrium's approach that starts with map-stage placement (termed *forward* here), the alternative (termed *reverse*) starts with the reduce-stage as follows: (i) assign reduce-tasks to each site in proportion to the slot distribution ($r_x = \frac{S_x}{\sum_x S_x}$); (ii) using this placement, solve the reduce task placement LP (from §3.2), but with the intermediate data fraction at each site now being our decision variables, giving us a desired intermediate data distribution (I_x^{shufl}) at each site x; (iii) solve the map task placement LP (from §3.1) but with an additional constraint of the intermediate data distribution, namely: $\sum_x m_{x,y} = \frac{I_y^{shufl}}{\sum_y I_y^{shufl}}$.

In our evaluations (§6) we compute both, *forward* and *reverse* solutions, and choose the better of the two. The results illustrate that we can obtain occasional benefits from this approach, but the overall improvements are marginal as compared to Tetrium. Given the small loss in performance and the fact that Tetrium's *forward* approach is easier to implement – it does not require upfront information about all stages and potentially incurs less overhead – our prototype implementation and simulations are focused on the originally proposed solutions (in §3.1 and §3.2).

4 Job Scheduling

Resource allocation among jobs is critical in reducing response time as illustrated in §2, however, the problem

of scheduling jobs with parallel tasks to minimize response time is NP-hard [24, 52]. We develop an efficient job scheduling heuristic for reducing the average job response time (§4.1) by incorporating the task placement and ordering approach mentioned earlier; we discuss the optimality of the solution in §8. We also provide flexibility to incorporate other important metrics (WAN usage in §4.3, fairness in §4.4) using simple knobs.

4.1 Minimizing Average Job Response Time

When it comes to reducing average job response time, it is intuitive to apply the the Shortest Remaining Processing Time (SRPT) philosophy due to its well-studied behavior. A key component in SRPT is the estimation of remaining time for each job, and previous works often resort to the number of remaining tasks as an approximation[48]. In geo-distributed clusters, however, the response time of a job is determined by not only its remaining tasks, but also how the tasks are distributed across the sites based on resource availability as highlighted by our illustrative examples (§2). We propose the following heuristic for estimating the remaining time for a geo-distributed job:

Remaining duration for jobs with stage-dependency Conventional jobs can be modeled as a DAG of tasks, in which there is stage dependency. Note that previous works [47, 48] often handle the DAG by treating each stage separately; yet this has the undesirable property of mistakenly allocating slots to quickly finish a stage with a lot of subsequent stages, while there could be stages from other jobs that are close to their overall completion.

Ideally, we should schedule the jobs based on their remaining time across all the stages. However, estimating such information across all the stages is inefficient to compute because we need to sequentially estimate each stage's processing time (by invoking the optimizer from §3) based on the output of the parent stages and sum them up.

Our simple heuristic first chooses the job's remaining number of stages (G^j) as a proxy for j's remaining workloads, then uses the current stage's remaining processing time (T^j) to break ties if there are multiple jobs with the same DAG progress. Once we identify job k with the shortest remaining time based on G^j and T^j, we allocate slots $D^k = \{d_x^k\}$ to job k based on the task placement described in §3 and schedule it to run.The algorithm continues the above steps until there is no remaining slot to be allocated, or all the jobs have been scheduled, whichever comes first.

4.2 Dealing with Resource Dynamics

Resource capacity at a site may suddenly drop due to various reasons: the compute slots at a site may be allocated to other non-analytics jobs with higher priority, and available bandwidth between the sites could degrade due to temporal link failure in WAN. The reduced resources at a site result in longer finish time for its assigned tasks, which could prolong the job's response time. Therefore, the global manager should adjust the workloads assignment, e.g., by offloading workloads from the site with resource drop, and update all the site managers. However, updating the assignment at all of the sites incurs significant overhead in communication. It is hence desirable to update only a subset of the sites, while still optimizing for job response time.

We extend our solution to address such resource dynamics. Let f_i denote the number of tasks of a job assigned to site i according to the last scheduling decision. When the global manger is notified of significant resource drop changes at a site by the site manager, it triggers the scheduling computation based on the new resource condition, and obtains the new assignment f_i^* for all sites. Assume the global manager is set to only update k sites; when k equals to the total number of sites, it gets the optimal assignment f_i^* for all site. Say after re-assignment, each site now has f_i' tasks, and we calculate a distance metric Q between this assignment and the ideal assignment: $Q = \sqrt{\forall_i f_i' - f_i^{*2}}$. The objective is to adjust the assignment in only k sites and find the new assignment with the minimum Q value. We design a heuristic towards this goal. We first focus on the sites that want to offload some of their tasks to other sits, i.e., $f_z^* - f_z < 0$ on site z, and sort the sites based on $|f_z^* - f_z|$ in descending order. We start moving tasks out of the first site to the other sites until $f_z' = f_z^*$. We exhaustively search through all possible assignments and update based on the one with the minimum Q value. In §6.3.2 we evaluate the performance of this approach under different k values.

4.3 Considering WAN Usage

WAN usage across the sites is critical for operational cost in global data analytics [55], and is often charged based on the volume of data transferred over WAN [2]. Limiting the amount of data sent across the sites, however, may increase a job's response time, because it restricts the job from transferring the data to a site with high resource capacity that can process the data faster.

Tetrium offers a knob, ρ, that incorporates budgeting WAN usage with reducing job response time. During each scheduling instance, Tetrium calculates WAN budget $W^j = W^j_{min} + \rho W^j_{max} - W^j_{min}$ for each job, where W^j_{max} and W^j_{min} are the maximum and minimum possible

WAN usage for the job, respectively. When $\rho \to 1$, a job has maximum WAN budget, and Tetrium is completely geared toward reducing the job's response time. On the other hand, as $\rho \to 0$, WAN usage is minimized for each job.

We set W^j_{max} to be the sum of input data in the job's current stage, as the amount of data this job could sent across WAN is no more than its input data. W^j_{min} value varies depending on the stage type: in a map stage, $W^j_{min} = 0$ as a job achieves zero data transfer when it leaves all input data in-place, while in a reduce stage, W^j_{min} can be calculated by the following LP model:

$$min \quad W^j_{min} \tag{11}$$

$$s.t. \quad W^j_{min} = {}_x I^{shufl}_x \cdot 1 - r_x \tag{12}$$

$$r_x \geq 0, {}_x r_x = 1, \forall x \tag{13}$$

, where the sum of upload data is constrained by W^j_{min}.[5]

Given the WAN budget calculated above, an additional constraint for data upload/download is then added in task placement model described in Section §3. In map stage, this constraint is written as ${}_x I^{input} \cdot {}_{y \neq x} m_{x,y} \leq W^j$; in reduce stage, it is ${}_x I^{shufl}_x \cdot 1 - r_x \leq W^j$. The revised models limit the amount of a job's data sent across sites when placing its tasks in each scheduling decision, and the rest follows Tetrium's original design.

4.4 Incorporating Fairness

Reducing average job response time based on SRPT may starve large jobs. Here we define fairness as: each job is allocated a number of slots in proportion to the number of its remaining tasks, i.e., job i with f_i remaining tasks gets $S^* \cdot \frac{f_i}{{}_i f_i}$ slots, where there are S^* available slots.

We provide some flexibility between reducing average job response time and achieving fairness: our system achieves ε-fairness if each job receives at least $1 - \varepsilon \cdot S^* \cdot \frac{f_i}{{}_i f_i}$ slots in each scheduling instance. The system achieves complete fairness as $\varepsilon \to 0$, while it reverts to the original design (geared towards performance) as $\varepsilon \to 1$.

Specifically, we first calculate the minimum slots that should be reserved for each job i as $p_i = S^* \cdot \frac{f_i}{{}_i f_i}$. Next, instead of allocating the selected job k all its desired slots D^k based on task placement, we limit the number of slots allowed for job k to $q_k = {}_x S_x - {}_{i \in J, i \neq k} p_i$. We scale down job k's slot allocation by $d^k_x \cdot \frac{q_k}{{}_x d^k_x}$ if $q_k < {}_x d^k_x$. After capping job k's slots based on $\min q_k, {}_x d^k_x$, the rest follows our original design.

[5]The sum of download data is equivalent to the sum of upload data, so specifying one of them is sufficient.

5 Prototype Implementation

We implement Tetrium on top of Spark [5] by overriding Spark's cluster scheduler to make our scheduling decisions, containing 950 LoC in Scala. In addition, our Spark implementation calls out Gurobi solver [6] to solve the models described in §3. The optimization model solving part contains roughly 300 LoC in Python. We discuss several implementation details as follows.

Batching of Slots: Since we make task placement and scheduling decisions based on currently available slots at one scheduling instance, the scheduling quality depends on whether the current set of available slots is sufficiently representative of future slot arrivals. We can make better placement decisions if we delay a bit so that more slots become available and more placement options can be covered, yet we do not want to waste resources by leaving them too long. In our implementation, we batch the slots according to the average duration of the recently finished tasks, which provides the system with a rich set of slots across sites so that the scheduler does not make biased decisions based on one (or a few) available slots.

Handling Dynamic Slot Arrivals: Slot distribution varies across scheduling instances in practice (even when batching is employed), either due to variance in last wave's task durations, or due to some slots being allocated to another jobs with higher priority. With such dynamic slot availability, our task ordering solution (§3.3) that schedules remote tasks prior to local tasks may force local tasks to run on remote sites given insufficient local slots in the end, which not only prolong the tasks' durations but also decreases slot utilization. To address dynamic slots availability in practice, Tetrium reserves a small fraction of the current slots for running local tasks, while leverages the remaining portion of the slots in accordance to the original task ordering method.

Estimation of Available Bandwidth: Given EC2 bandwidth is relatively stable at minutes level, similar to [47], we run measurements of available bandwidth at each site every few minutes. We do not apply specific bandwidth reservation method and assume that available bandwidth is fairly shared among all concurrent flows at a site.

Estimation of Task Duration: Our implementation estimates the tasks' duration according to the finished tasks in the same stage. Previous work [15] showed this approach is effective because tasks in the same stage perform the same functions over similar amounts of data.

6 Evaluation

We evaluate Tetrium with a geo-distributed EC2 deployment (§6.2) and extended trace-driven simulations (§6.3).

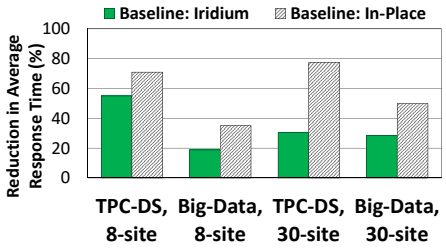


Figure 5: Reduction in Average Response Time. (Tetrium**'s average response time is** 75 **sec.)**

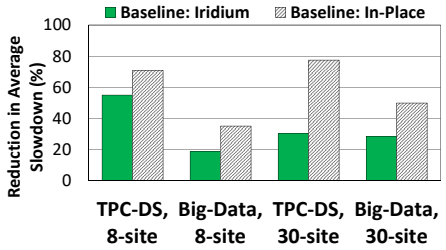


Figure 6: Reduction in Average Slowdown.

6.1 Settings

EC2 Deployment: We deploy Tetrium in EC2 using geo-distributed instances from 8 regions: Oregon (US), Virginia (US), Sao Paulo, Frankfurt, Ireland, Tokyo, Sydney and Singapore [2]. Our cluster allocates one EC2 instance in each region, and has heterogeneous compute capacities across the regions: the maximum slot number is 16 (*c4.4xlarge*), and the minimum is 4 (*c4.xlarge*); the bandwidth ranges from 100$Mbps$ to 1$Gbps$ between the instances. We also mimic a 30-site deployment by allocating 30 instances within one region.

Trace-driven Simulator: We evaluate Tetrium in large-scale settings and extended experiment duration using trace-driven simulations. The trace is derived from a production cluster with information including job arrivals, jobs' number of tasks and corresponding DAG, input/output data sizes for each task, the distribution of input data, straggler tasks and fail-over [13, 16]. We simulate a 50-site setting: the number of slots at each site ranges from 25 to 5000, for a mix of powerful datacenters and small edge clusters, with bandwidth ranging from 100$Mbps$ to 2$Gbps$.

Baselines: We use two baselines: (a) *In-Place Approach:* This is the default Spark [5] that runs tasks locally along with their input data (site-locality). It applies fair scheduling among the jobs and delay scheduling [56] for launching tasks within a job. (b) *Iridium [47]:* This is a recent work that improves Spark through shuffle-optimized reduce-tasks placement for geo-distributed jobs. Additional baselines are included as appropriate.

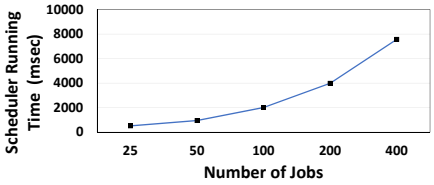


Figure 7: Running Time of Scheduling Decisions.

Performance Metrics: The primary performance metric in our evaluation is *average job response time*, and we report the results based on the reduction in response time as compared to different baselines. In some results we also report the reduction in *slowdown*. Slowdown is defined as the job's response time divided by its service time when running in isolation, and indicates how jobs are prioritized compared to their size.

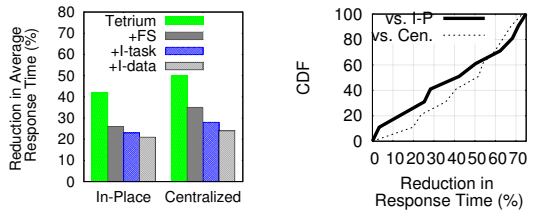
The results are obtained in settings where ρ (WAN budget) and ε (fairness) are set to 1, i.e., only focusing on reducing response time, if not explicitly specified.

6.2 Evaluation with EC2 Deployment

Workloads: We run two workloads to evaluate Tetrium in the EC2 deployment. (a) *TPC-DS* [8]: a set of SQL queries for evaluating decision support solutions, including Big Data systems. The queries in this benchmark are characterized by high CPU and I/O workloads, and typically contain long sequences of dependent stages ($6 \sim 16$). (b) *BigData* [3]: a mix of scan, join, and aggregation queries over the data provided in [46]; the queries in this benchmark have fewer sequences of stages ($2 \sim 5$).

Performance Gains: Figure 5 shows that Tetrium improves job average response time, as compared to In-Place and Iridium, by up to 78% and 55%, respectively, as a result of efficient task placement that jointly considers both network and compute resources. Tetrium's gains are also due to being able to schedule the jobs finishing faster so that they are not substantially delayed by longer jobs.

The performance gains are more significant under TPC-DS workloads than under Big Data Benchmark, and we believe this attributes to the job characteristics in these benchmarks: TPC-DS jobs have longer sequence of stages, which requires more task placement decisions and provides opportunity for larger gains. Gains under 30-site setting are more significant than in 8-site setting because Tetrium benefits more from the flexibility in placement options. Yet, Tetrium's gains compared to Iridium in "TPC-DS 30-site" are smaller than that in 8-site setting, because Iridium's network-centric approach also gains some benefits from the increased placement flexibility, especially in the TPC-DS workloads that incur lots of intermediate data shuffle during job execution.



(a) Average Response Time (b) CDF of Response Time

Figure 8: Response Time Reduction Compared to Baselines

Figure 6 shows Tetrium achieves up to 45% and 16% reduction in slowdown compared to In-Place and Iridium, respectively. Tetrium has the best slowdown, as it not only reduces a job's response time by efficient task placement, but also prioritizes small jobs (to finish quickly) without being delayed by large jobs. As expected, In-Place has the worst slowdown, as it essentially allocates slots based on fair sharing among jobs. Tetrium has less gains compared to Iridium as Iridium achieves better slowdown (compared to In-Place) by reducing response time through better (network-centric) task placement.

Scheduling Overhead: Figure 7 shows Tetrium's scheduling overhead scales well as the number of jobs increases: Tetrium's scheduling decisions complete within $\approx 950ms$ for 50 concurrent jobs and $\approx 8s$ for 400 jobs. The majority of scheduling time is spent on task placement decisions, which takes an average of $100ms$ for each decision for a job. Although the number of jobs in the system might be large, Tetrium effectively selects a few high-priority jobs (by focusing on the jobs with fewer remaining stages) and eliminates the needs for solving optimization problems for lower priority jobs. This leads to a significant reduction in scheduling overhead and hence the scalability of the proposed scheduling algorithm. In practice, solving task placement model for each job can run in parallel and further reduces the scheduler's running time.

6.3 Evaluation with Trace-driven Simulations

Our simulations employ an additional baseline (*Centralized Approach*) that (upfront) aggregates all input data at a powerful datacenter, to quantify the benefits in running tasks of the jobs across the site.

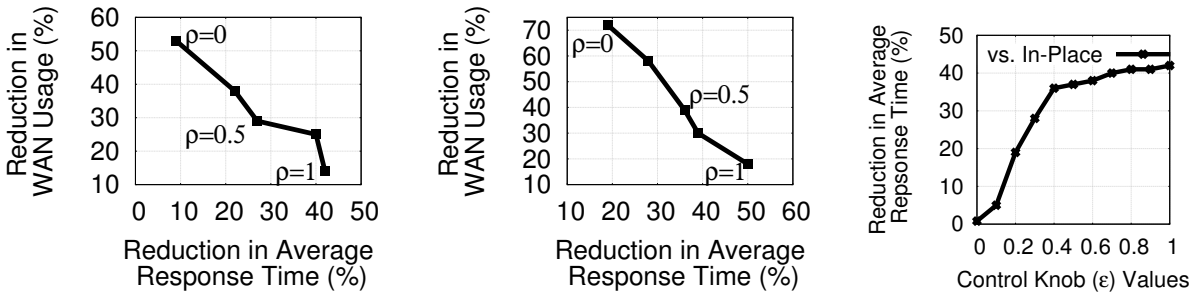
6.3.1 Performance Gains and Design Choices Reducing Response Time: In Figure 8(a), Tetrium achieves 42% and 50% improvements as compared to In-Place and Centralized, respectively. The gains are higher against Centralized because most jobs have less intermediate data than input data, which undermines the benefits in pre-aggregating input data upfront. Next, we tease apart

Map	Remote-Spread	Remote-Spread	Local-First	Local-First
Reduce	Local-First	Random	Local-First	Random
Gains	42%	38%	32%	29%

Figure 9: Gains in Response Time Under Different Combinations of Task Ordering Strategies (Baseline: In-Place)

Tetrium's gains by quantifying the contributions of the task placement and job scheduling strategies using: (a) Tetrium *with Fair Scheduler (*Tetrium+*FS)* which replaces Tetrium's job scheduling method by fair scheduling. (b) Tetrium *with Iridium's task placement (+I-task)* which replaces Tetrium's task placement method by that of Iridium's. (c) Tetrium *with Iridium's data placement (+I-data)* that applies Iridium's data placement method to original Tetrium. Tetrium+*FS* also provides significant gains (of 26% and 35%), which verifies that Tetrium's task placement solution is effective even without the aid of the job scheduling solution. Also note that moving data in advance (Tetrium+*I-data*) does not help Tetrium as it is difficult to predict the resource availability in future scheduling instances. Although not included in the figure, Tetrium also improves Tetris [28] by 33% (in average) and 47% (at 90^{th}-percentile) because Tetrium does not rely on pre-determined resource requirement of jobs as in Tetris and can speed up a job's response time by allocating more bandwidth if beneficial. Figure 8(b) plots the CDF of reduction of jobs' response times against In-Place and Centralized. Tetrium does not slow down any jobs compared to both baselines despite its greedy heuristic.

Task Ordering Strategy: We evaluate different task ordering strategies. For map-stage: (a) *Remote-First (Spread)*: launching remote tasks first while spreading them across different sites to reduce network contention, as proposed in §3.3, and (b) *Local-First:* launching local tasks first, i.e., those that read data from the site corresponding to the available slot. For reduce-stage: (a) *Longest-First:* first launching the reduce-task with the longest network transfer time, as proposed in §3.3, and (b) *Random:* arbitrarily selecting a reduce-task to run. Figure 9 presents the average gains for the 4 combinations of the strategies, as compared to In-Place baseline. The results verify that our proposed task ordering methods result in the best combination, as remote tasks are indeed the bottleneck among the tasks associated with the same stage. Therefore starting the remote tasks first rather than leaving them in the end delaying the completion time reduces the response time. We also notice that most of the gains are attributed to the map-task ordering method. This is because reduce tasks need to read data from many (potentially all) sites, its completion depend more on how reduce tasks are placed than their ordering.



(a) Baseline: In-Place (b) Baseline: Centralized (c) Baseline: In-Place

Figure 10: Balancing Response Time And Other Performance Goals: (a)(b) Reducing Response Time With Budget Of WAN Usage. (c) Recuing Response Time vs. Achieving Fairness.

Resource Drop (%)	k = 3	k = 5	k = 7	k = 10	k = 20	k = 50
10	22	35	37	38	39	39
20	18	29	33	34	35	35
30	16	25	26	26	32	34
40	9	19	23	25	26	30
50	6	15	18	20	21	24

Figure 11: Gains in Response Time Under Different Resource Dynamics Scenarios (Baseline: In-Place)

Stage-By-Stage Approach: We investigate the effectiveness of Tetrium's solution to address the mismatch between map and reduce task placement (§3.4). We quantify this limitation by comparing Tetrium's (*forward*) stage-by-stage approach against a method that selects the best out of *forward* and *reverse*, while the latter is guaranteed to be better (or at least as good as) Tetrium's approach. Our results show that Tetrium achieves 42% improvements against In-Place baseline, while the mixed method achieves 45%. Because (i) the difference is marginal and (ii) *forward* is more practical to implement in most systems while incurring less overhead (e.g., it does not require upfront information about all stages), we adopt *forward* stage-by-stage as in Tetrium. Another subtle point is that, in most data analytics jobs, the volume of intermediate data quickly drops in the subsequent stages, which mitigates the impacts of stage-by-stage's non-optimal placement in the later stages.

6.3.2 Addressing Resource Dynamics We next evaluate the our proposed method for addressing resource dynamics (§4.2). In this experiment there are 50 sites, and during the job running we degrade both the compute and network resources of 5 randomly selected sites by a certain percentage. Figure 11 presents the gains compared to In-Place baseline under various resource drops (%) as well as the number of sites allowed for updates (k). For example, when the resource drops by 30% and the system is allowed to update 7 sites, the proposed method achieves 26% reduction in job response time.

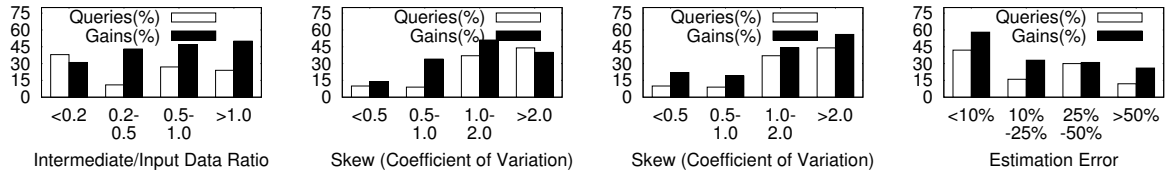
Gains increase when the system is allowed to update more sites. On the other hand, when the resources drop by a larger percentage, the gains decrease. Under a setting of 50 sites, setting $k = 5$ or 7 provide most of the gains, while setting k beyond 10 does not improve further.

6.3.3 Incorporating WAN Usage Budget Figure 10 presents the reduction in response time and WAN usage, compared to In-Place [6] and Centralized baselines, under various knob ρ values. As ρ increases, more WAN budget is allowed, and hence the reduction in response time significantly increases. Tetrium saves more WAN usage compared to Centralized than In-Place. This is because In-Place does save some WAN usage by not transferring any data in map-stage, while Centralized aggregates all input data upfront. Tetrium saves 53% WAN usage given the tightest WAN budget ($\rho = 0$), and achieves at least 14% reduction even when fully geared toward reducing response time ($\rho = 1$). In general, WAN budget has a linear impact on reduction in response time. We believe this is because restricting to less WAN usage forces more tasks to run locally, and the running time is roughly linear to the amount of workloads required. The results also suggest a sweet spot at $\rho = 0.75$, where Tetrium achieves 40% reduction in response time while reducing 25% WAN usage.

In our experiments, 36% of data transfer across sites come from the map-stages, while the other 64% come from reduce-stages. We also notice that the majority of data transfer take place during the first few stages, in which multiple tables of data are retrieved and joined in this process, while the volume quickly decreases after then because the size of intermediate data usually drops quickly in data analytics jobs.

6.3.4 Balancing Response Time and Fairness Figure 10(c) shows gains (i.e., reduction in response time)

[6] Note that In-Place does incur network transfer in intermediate stages (e.g., reduce) although it processed all input data locally in map-stage.



(a) Ratio of Intermediate/Input Size (b) Input Data Skew (c) Intermediate Data Skew (d) Task Estimation Error

Figure 12: Distribution of The Gains under Various Categories.

under various fairness knob's (ε) values. Under complete fairness ($\varepsilon = 0$), Tetrium achieves similar performance as In-Place which adopts fair sharing among the jobs. Although Tetrium assigns slots according to each job's desired placement distribution, the number of slots allocated under complete fairness is not enough for each job to take the advantage of efficient placement decisions. As the knob is turned towards $\varepsilon = 1$ (i.e., response time oriented design), gains increase quickly because – in addition to providing each job some minimum number of slots – Tetrium allocates slots in proportional to each job's desired distribution, thereby achieving a good match between available resources and demand. Evaluation with our workloads suggests a sweet spot around $\varepsilon = 0.6$, where Tetrium reserves 40% slots in proportional to jobs' sizes while achieving 38% reduction in response time.

6.4 Distribution of The Performance Gains

We further study Tetrium's gains (compared to the In-place baseline) by characterizing the gains distribution.

Intermediate-Input Ratio: Figure 12(a) shows the gains based on intermediate-input ratio of the jobs: the higher the ratio, the more improvements (up to 50%) Tetrium achieves. At the high end (reduce-heavy), more intermediate data is generated than input data, which incurs more data shuffle across the network. Therefore Tetrium benefits more from effective resource allocation. At the low end (map-heavy), Tetrium also improves by at least 31%; this attributes to effective map-task placement, which also highlights how site-locality falls short.

Data Skew: Tetrium's gains depend on the skews in data distribution – Coefficient of Variation (CV) – across sites. Up to $CV \leq 2.0$ in Figure 12(b), higher data skews in input data benefit greatly from better decisions by Tetrium in balancing computation and network resources. However, when the input data is extremely skewed ($CV > 2.0$), it resides on only a few site (one site at the extreme), and In-Place benefits from reduced network transfer time for later stages through site-locality. Figure 12(c) depicts effects of intermediate data skew, where gains are highest (as high as 56%) at the most skewed data distributions.

Task Estimation Error: Figure 12(d) shows that Tetrium achieves the highest gains with accurate estimation in task duration. Although the gains drops when estimation error increases, only a small percentage of the tasks have high estimation error in our implementation.

Heterogeneity of Resources: We evaluate the impact of skew of slots and bandwidth by setting it based on Zipf distribution: the higher the exponent e value, the more skewed the resources to a few sites. Settings with more skewed resources provide more gains, as such scenarios calls for wiser task placements by balancing the workloads according to resource availability. Results suggest that the compute slot skew has more impact than the bandwidth skew: from slot skew $e = 0$ to 1.6, the gain increases by 51%, while from BW skew 0 to 1.6, the gain increase by 37%. This is because the number of slots *directly* determines the compute duration once the placement is set.

7 Related Work

Multi-resource Scheduling: Previous works [26, 27] achieve fairness by allocating only compute resources (CPU and memory). However, for geo-distributed jobs, network resource is also critical for performance.

Other works [28–30] pack multiple resources (CPU, memory, bandwidth) for the tasks based on pre-configured resource requirement to avoid resource fragmentation. In heterogeneous geo-distributed data analytics, the network transfer time portion of a job's duration depends on how the tasks are distributed across sites, as well as the bottleneck, given the heterogeneous nature of bandwidth. Pre-configuring a static bandwidth requirement for each task upfront as in those works does not capture such nature of the heterogeneous geo-distributed clusters because network bandwidth is inherently a fungible resource. Specifically, a task should be able to use whatever available bandwidth at the time as opposed to use only an isolated fraction. Tetrium, on the other hand, schedules tasks based on equal-sized computation resources (i.e., static combination of CPU and memory) while allows tasks to utilize as much network bandwidth at the time as possible.

Geo-distributed Jobs: Many works [31, 39, 47, 54, 55] focus on reducing network transfer delay of geo-distributed jobs, while SWAG [32] focuses on allocating compute slots through coordinated job scheduling. Tetrium considers both network and computation resources, and is the first to address the inter-dependency of task placement and job scheduling in the geo-distributed setting where resources are heterogeneous and constrained.

Intra-cluster Jobs: Prior work on reducing network delay of tasks focus on data locality [34, 56] or constraining the tasks to a few racks [36].There also exist many works [20–23] for reducing the response time of *coflows*, i.e., a set of flows that belong to the same job. None of these, however, consider the resource heterogeneity and data skew across sites. Our work is the first to address multi-resource allocation in geo-distributed analytics.

Improving performance by optimizing job/query execution plan has been an active research area [11, 50, 51, 54]. Those works computes the structure of the job (e.g., order of join or partition strategy) and generate the job execution plan (i.e., DAG). Tetrium takes the job execution plans generated by those mechanisms as input, then places tasks and schedules jobs according to Tetrium's scheduling heuristic to reduce jobs' response times.

Job Scheduling: Theoretical work in concurrent open shops [25, 43, 49] propose approximations for allocating compute slots among jobs across machines. Other theory works [19, 33, 44] focus on scheduling compute slots for (and placing) tasks with precedence constraints within a job. Those theory works require certain assumptions (e.g., knowing exact task duration) that can not work in practice. In addition, network transfer time depends on how all the concurrent tasks are placed across the sites in practice, instead of a fixed delay for each source-destination pair as modeled in [44]. To the best of our knowledge, addressing both job scheduling and task placement with multi-resource consideration has not been well studied yet.

8 Discussion

Theoretical Performance Guarantee. This paper addresses a new challenge of resource scheduling in geo-distributed settings.In this work we focus on the system perspectives of the problem by proposing an efficient heuristic and addressing several practical issues in real-world deployment to make it more robust. We acknowledge that our proposed heuristic is based on intuitive design principles and modeling with simplified assumptions, which has no theoretical guarantee in performance. Understanding the optimality of our heuristic, along with developing approximation with performance guarantee, is a promising direction of the future work.

Joint-Stage Task Placement. Scheduling the tasks by considering the job's full DAG has been an open problem, and previous works treat the tasks of each stage separately[39, 47, 48]. While we acknowledge that our proposed solution is built based on the simplified problem as previous works do (i.e., stage-by-stage planning), we made a complimentary attempt for improving our proposed stage-by-stage approach (Section §3.4), and our evaluation shows that such improvement is marginal. The dynamics of available resources and concurrent workloads in the systems can be one of the reasons that joint-stage task placement might not get significant performance improvement as expected. Designing a more sophisticated solution based on joint-stage task placement based on the above mentioned factors is a challenging future work.

Incorporating Other Scheduling Decisions. In addition to the challenges addressed in this paper (i.e., task placement and ordering, job scheduling), there exist other important factors that can be incorporated into our resource scheduling decisions: (a) Our solution is based on the assumption that there is only single primary item for each data. Our task placement model (Section §3) can be extended to consider the selection from multiple data replica by introducing additional variables and constraints. (b) Stragglers are common in large-scale data-parallel jobs. Launching redundant copies of stragglers is the mainstream approach of mitigating the performance degradation [48, 57], and is orthogonal to our solution. One can trade off launching original tasks for redundant copies associated with the same job or even across the jobs, while placing the redundant copies across the sites using the proposed heuristic in this paper.

9 Conclusions

As cloud providers deploy datacenters around the world, support for fast geo-distributed data analytics is becoming critical. We design and build Tetrium, a system for multi-resource allocation in geo-distributed clusters, that takes the first stab at jointly scheduling the network and computation resources for improving geo-distributed jobs' response times, while it incorporates other metrics (e.g., WAN usage, fairness or makespan) with simple knobs. In our evaluations with a geo-distributed EC2 deployment and large-scale trace-driven simulations, Tetrium greatly improves the average job response time compared to common practices and recent approaches.

Acknowledge

We would like to thank the anonymous reviewers and our shepherd, Dejan Kostic, for their thoughtful suggestions. This work is supported in part by the Zumberge Research Award.

References

[1] Cloud vs edge in an IoT world. https://iotworldnews.com/2016/04/cloud-vs-edge-in-an-iot-world/.

[2] http://aws.amazon.com/about-aws/global-infrastructure/. Amazonn Global Infrastructure.

[3] https://amplab.cs.berkeley.edu/benchmark/. AMPLab Big Data Benchmark.

[4] https://ipp.mit.edu/sites/default/files/documents/congestion-handout-final.pdf, 2014. Measureing Internet Congestion: A preliminary report.

[5] http://spark.apache.org/. Spark Cluster Computing System.

[6] http://www.gurobi.com/. Gurobi Optimization.

[7] http://www.microsoft.com/en-us/server-cloud/cloud-os/global-datacenters.aspx. Microsoft Cloud Platform.

[8] http://www.tpc.org/tpcds/. TPC Benchmark Standard for Decision Support Solutions Including Big Data.

[9] Key azure storsimple features. https://www.microsoft.com/en-us/server-cloud/products/storsimple/Features.aspx.

[10] Private conversation with datacenter operators of one of the largest public cloud providers; anonymized. 2016.

[11] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology (EDBT)*, 2010.

[12] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: Coping with skewed content popularity in mapreduce clusters. In *Proceedings of the Sixth Conference on Computer Systems (EuroSys)*, 2011.

[13] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2013.

[14] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. Pacman: Coordinated memory caching for parallel jobs. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2012.

[15] G. Ananthanarayanan, C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu. Grass: trimming stragglers in approximation analytics. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2014.

[16] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010.

[17] B. G. K. K. Ashish Vulimiri, Carlo Curino and G. Varghese. Wanalytics: Analytics for a geo-distributed data-intensive world. In *CIDR*, 2015.

[18] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan. Mapping the expansion of google's serving infrastructure. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC)*, 2013.

[19] C. Chekuri and M. Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 2001.

[20] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[21] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2011.

[22] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2014.

[23] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2014.

[24] J. Du and J. Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM J. Discret. Math.*, 1989.

[25] N. Garg, A. Kumar, and V. Pandit. Order scheduling models: hardness and algorithms. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*. Springer, 2007.

[26] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2011.

[27] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Choosy: Max-min fair sharing for datacenter jobs with constraints. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*, 2013.

[28] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2014.

[29] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan. Altruistic scheduling in multi-resource clusters. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2016.

[30] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni. Graphene: Packing and dependency-aware scheduling for data-parallel clusters. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2016.

[31] 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 *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (NSDI)*, 2017.

[32] C.-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.

[33] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal of Computing*, 1989.

[34] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

[35] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2013.

[36] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar. Network-aware scheduling for data-parallel jobs: Plan when you can. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[37] J. Jiang, R. Das, G. Ananthanarayanan, P. A. Chou, V. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin, and H. Zhang. Via: Improving internet telephony call quality using predictive relay selection. In *Proceedings of the*

ACM Conference on Special Interest Group on Data Communication (SIGCOMM), 2016.

[38] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[39] K. Kloudas, M. Mamede, N. Preguica, and R. Rodrigues. Pixida: Optimizing Data Parallel Jobs in Wide-Area Data Analytics. In *International Conference on Very Large Data Bases (VLDB)*, 2015.

[40] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2011.

[41] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2012.

[42] H. H. Liu, R. Viswanathan, M. Calder, A. Akella, R. Mahajan, J. Padhye, and M. Zhang. Efficiently delivering online services over integrated infrastructure. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[43] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operation Research Letter*, 2010.

[44] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós. On scheduling in map-reduce and flow-shops. In *Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2011.

[45] K. Ousterhout, A. Panda, J. Rosen, S. Venkataraman, R. Xin, S. Ratnasamy, S. Shenker, and I. Stoica. The case for tiny tasks in compute clusters. In *Presented as part of the 14th Workshop on Hot Topics in Operating Systems (HotOS)*, 2013.

[46] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 2009.

[47] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica. Low latency geo-distributed data analytics. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[48] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[49] T. A. Roemer. A note on the complexity of the concurrent open shop problem. Springer, 2006.

[50] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *ACM SIGMOD Record*, 2000.

[51] T. Sellis and S. Ghosh. On the multiple-query optimization problem. *IEEE Transactions on Knowledge and Data Engineering*, 1990.

[52] J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms scheduling parallelizable tasks. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, 1992.

[53] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. Franklin, and I. Stoica. The Power of Choice in Data-Aware Cluster Scheduling. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2014.

[54] R. Viswanathan, G. Ananthanarayanan, and A. Akella. Clarinet: Wan-aware optimization for analytics queries. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2016.

[55] 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 *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2015.

[56] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *ACM EuroSys*, 2010.

[57] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.