Tempo: Robust and Self-Tuning Resource Management in Multi-tenant Parallel Databases [*]

Zilong Tan
Duke University
ztan@cs.duke.edu

Shivnath Babu
Duke University
shivnath@cs.duke.edu

ABSTRACT

Multi-tenant database systems have a component called the *Resource Manager, or RM* that is responsible for allocating resources to tenants. RMs today do not provide direct support for performance objectives such as: "Average job response time of tenant A must be less than two minutes", or "No more than 5% of tenant B's jobs can miss the deadline of 1 hour." Thus, DBAs have to tinker with the RM's low-level configuration settings to meet such objectives. We propose a framework called *Tempo* that brings *simplicity*, *self-tuning*, and *robustness* to existing RMs. Tempo provides a simple interface for DBAs to specify performance objectives declaratively, and optimizes the RM configuration settings to meet these objectives. Tempo has a solid theoretical foundation which gives key robustness guarantees. We report experiments done on Tempo using production traces of data-processing workloads from companies such as Facebook and Cloudera. These experiments demonstrate significant improvements in meeting desired performance objectives over RM configuration settings specified by human experts.

1. INTRODUCTION

Many enterprises today run multi-tenant database systems on large shared-nothing clusters. Examples of such systems include parallel SQL database systems like RedShift [1], Teradata [5], and Vertica [6], Hadoop/YARN running SQL and MapReduce workloads, Spark running on Mesos [25] or YARN [46], and many others. Meeting the performance goals of business-critical workloads (popularly called *service-level objectives*, or *SLOs*) while achieving high resource utilization in multi-tenant database systems has become more important and challenging than ever.

The problem of handling many (often in 1000s) small and independent databases on a multi-tenant database Platform-as-a-Service (usually called *PaaS* or *DBaaS*) has received considerable attention in recent years [50, 37, 32, 36, 15]. That is not the problem we focus on in this paper. Our focus is on handling fewer, but much "bigger", tenants who process very large amounts of data on a shared-nothing cluster that is usually run within an enterprise. Hadoop, Spark, Teradata, Vertica, etc., are typically run in such settings.

These multi-tenant database systems each have a component—commonly referred to as the *Resource Manager (RM)*—that is responsible for allocating resources to tenants. Most widely deployed RMs like YARN and Mesos focus on resource isolation and do not support SLOs. Instead, they rely on the Database Administrator (DBA) to "guesstimate" answers to questions such as: "How much resources are needed to complete this job before its deadline?" Then, DBAs have to translate their answers into low-level configuration settings in the RM. This process is brittle and increasingly hard as workloads evolve, data and cluster sizes change, and new workloads are added. Thus, techniques have been proposed in the literature to support specific SLOs such as deadlines [14, 33, 18, 47], fast job response times [11, 14, 21, 39], high resource utilization [2, 11, 14], scalability [2, 43, 51], and transparent failure recovery [51].

In this paper, we present a framework called *Tempo* that brings three properties to existing RMs: *simplicity*, *self-tuning*, and *robustness*. First, Tempo provides a simple interface for DBAs to specify SLOs declaratively. Thus, Tempo enables the RM to be made aware of SLOs such as: "Average job response time of tenant A must be less than two minutes", and "No more than 5% of tenant B's jobs can miss the deadline of 1 hour." Second, Tempo constantly monitors the SLO compliance in the database, and adaptively optimizes the RM configuration settings to maximize SLO compliance. Third, Tempo has a solid theoretical foundation which gives five critical robustness guarantees:

1) Tempo makes high-quality resource scheduling decisions in presence of noise, e.g., job failures, commonly observed in production database systems.
2) Tempo delivers provably end-to-end tenant performance isolation with Pareto-optimal SLOs. This is often more desirable than traditional resource isolation.
3) When all SLOs cannot be satisfied—which is common in busy database systems—Tempo guarantees max-min fairness over SLO satisfactions [34].
4) Tempo adapts to workload patterns and variations.
5) Tempo reduces the risk of major performance regression while being applied to production database systems.

We have implemented Tempo as a drop-in component in the RMs used by multi-tenant databases running on Hadoop and Spark. We report experiments done using production traces of data-processing workloads from companies such as Facebook and Cloudera. The experiments show that Tempo can reduce the average job response time by 50% for best-effort workloads and increase resource utilization by 15%, without hurting the deadline-driven workloads.

[*]This work was supported by grants CNS1423128, IIS1423124, CNS1218981, and IIS0964560.

Proceedings of the VLDB Endowment, Vol. 9, No. 10
Copyright 2016 VLDB Endowment 2150-8097/16/06.

Table 1: Tenant characteristics at Company ABC.

Tenant	Characteristics
BI	I/O-intensive SQL queries
DEV	Mixture of different types of jobs
APP	Small, lightweight jobs
STR	Hadoop streaming jobs
MV	Long-running, CPU-intensive
ETL	I/O-intensive, periodic but bursty

This paper makes the following contributions:

- We propose several robustness properties of RMs, and investigate ways to achieve robustness provably.
- We provide a solid theoretical foundation for multi-objective SLO optimization under uncertainty, and present a novel solution algorithm.
- We design time-warp based mechanisms to estimate the impact of different RM configurations on tenants' SLOs.
- We evaluate Tempo for various real-life use cases based on production traces.

2. BACKGROUND

Tempo's design was motivated by our observations from several large production database systems. While designing Tempo, we analyzed workload traces from three companies each of which runs multi-tenant database systems on large clusters. Two of these systems run on 600+ nodes while the other runs on about 150 nodes. (While all three are well-known companies, we cannot share their names due to legal restrictions.) We talked to business analysts, application developers, team managers, and DBAs in these teams to understand the SLOs that they need to meet and the challenges they face in resource management. From all our interviews, the following emerged as the top concerns:

- Concern A: Deadline-based workloads and best-effort workloads have to be supported on the same database system.
- Concern B: Repeatedly-run jobs often have unpredictable completion times.
- Concern C: Resource utilization was lower than expected.
- Concern D: Resource allocation does not adapt automatically to the patterns and variations in the workloads.

To elaborate on these four concerns, we will use one of the three companies—henceforth, referred to as Company ABC—which is a real-life company that runs a multi-tenant database system on a 700-node Hadoop cluster with over 30 Petabytes of data.

2.1 Concern A

Company ABC has three types of users who generate database workloads. Business Intelligence (BI) analysts and Data Scientists predominantly do exploratory analysis on the data. Engineers develop and maintain recurring jobs that run on the database. One such category of jobs is Extract-Transform-Load (ETL) which brings new data into the system. Each job goes through many runs in a development phase on the cluster before being certified to run as a production job. Thus, the system supports both development and production runs of jobs.

Distinct workloads from these users form the *tenants* in the multi-tenant system. Table 1 shows the six tenants at Company ABC and their distinct workload characteristics. (The experimental evaluation section gives more fine-grained details of these workloads.)

The BI and ETL users correspond directly to similarly-named tenants. Among the other tenants, MV corresponds to the creation of *Materialized Views* such as joined results of multiple tables as well as statistical models created from the incoming data brought through ETL. The BI users and Data Scientists usually write their queries and analysis programs on these materialized views. The APP tenant runs jobs from a specific high-priority production application. The DEV and STR tenants mostly comprise queries and analysis programs being run as part of application development by engineers and Data Scientists. At Company ABC:

- Jobs from the ETL and MV tenants have deadlines because any delay in these jobs will affect the entire daily operations of the company. We have seen multi-day delays caused by deadline misses for the ETL and MV tenants that had significant business impact.
- About 30% of high-priority jobs in APP miss deadlines.
- While all tenants want as low job response time as possible for completion of their jobs, BI, DEV, and STR are treated as "best-effort" tenants in that the goal is to provide their jobs as low response time as possible subject to meeting the requirements of the ETL, MV, and APP tenants.

2.2 Concern B

Predictability of completion time for recurring jobs is a key need in most companies. This demand stems from ease of resource planning and scheduling for dependent jobs. At Company ABC:

- The completion of one of the recurring jobs of the ETL tenant varies between 5 and 60 minutes.
- The completion of one of the recurring jobs of the MV tenant varies between 2 and 6 hours.

While we observed that this variance is caused partly by variation in the input sizes of the jobs across runs, these sizes exhibit strong temporal patterns. For example, the input sizes of the recurring jobs in ETL vary across days within a week, but remain stable across multiple weeks.

2.3 Concern C

Resources can be wasted in multi-tenant systems due to reasons such as: (i) task preemption; (ii) suboptimal configuration of resource limits; and (iii) jobs in poorly-written queries being killed by DBAs. At Company ABC, 17.5% of map tasks and 27.7% of reduce tasks were preempted for the jobs run by the MV tenant over a week interval. This caused considerable amount of wasted resources, especially because the reduce tasks of the MV tenant have long execution times.

2.4 Concern D

A resource allocation which meets the SLOs perfectly at one moment may be sub-optimal at another moment due to various factors. First, input data sizes for a tenant may vary considerably across shorter time intervals while showing distinct patterns across longer intervals. At Company ABC, ETL jobs process Web activity logs which come in much smaller quantities on weekends. Second, the resource demands of different tenants can be correlated over time. For example, Figure 1 shows the memory consumption of two tenants at Company ABC over the course of a day. The horizontal lines in the figure show the respective resource limits that have been configured by the DBA to protect against resource hoarding by tenants. Notice that while there are periods where both tenants use up all available resources, there are other periods where the configured resource limit prevents one tenant from using the resources unused by the other.

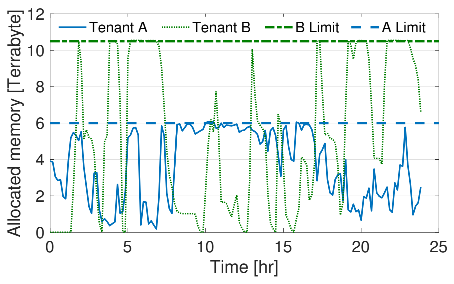

Figure 1: Memory consumption of two tenants during a day.

3. OVERVIEW OF PROBLEM

From our interviews, two salient points emerged that summarize the crux of what Tempo attempts to solve:

- Workloads in multi-tenant parallel databases have SLOs. Current RMs do not provide easy ways to ensure that these SLOs are satisfied.
- Current RMs require the DBA to estimate resources to meet the per-tenant SLOs, and then specify low-level RM configuration like resource shares, resource limits, and preemption timeouts in order to meet these SLOs. This process is brittle and increasingly hard as workloads evolve, data and cluster sizes change, and new workloads are added.

3.1 Multi-tenant Workloads

Parallel databases decompose queries and analysis programs to Directed Acyclic Graphs (DAGs) of jobs that each consist of one or more tasks. We will use the following representation for multi-tenant workloads throughout this paper:

Task: A task is the smallest unit of work for the purpose of resource allocation. A task has a *start time* when the task acquires resources to start running; and a *finish time* when the task completes. We consider workloads where the task *duration*, which is the interval from start time to finish time, is given.

Job: A job consists of a set of parallel and/or dependent tasks, e.g., map and reduce tasks in a MapReduce job or the tasks in a Spark stage. The job response time is the interval from the job *submission time* to the finish time of the last task. Job response times vary based on when resources are allocated to the tasks of the job.

Workload: A workload is a *fixed* sequence of jobs over a period of time where the submission time of each job and the tenant submitting the job are given.

Throughout this paper, we use w to denote a multi-tenant workload. In Tempo, w can be obtained in one of two ways. First, w can be an actual job trace submitted to a parallel database system. We use this approach in our experiments. Second, w can be generated from a statistical workload model as we will discuss later.

3.2 SLOs

From our interviews with users and DBAs, we identified five major classes of SLOs. Rather than aiming for any particular job, these SLOs reflect the performance of the workload as a whole over a period of time. Thus, Tempo deals with these workload-level SLOs that are defined over a given time window.

The first class specifies the fraction of jobs that violated the deadline. For recurring jobs, the deadline is either the start of the next run or an absolute time point like 5:00 AM. The second class specifies that the average job response time must be less than a given threshold. Such SLOs are often associated with ad-hoc jobs. The third class is about ensuring that each tenant gets a fair allocation of resources. In particular, when the database is under contention, the proportion of resources allocated to each tenant must adhere to predetermined values. This SLO class prevents individual tenants from monopolizing the resources intentionally or otherwise. Fourth, the resource utilization or job throughput must be above a threshold. This SLO class generally serves the interest of DBAs to maximize the return on investment (ROI) in the cluster. A fifth type of SLO orders the other SLOs in terms of priority. This special SLO mandates that SLOs with higher priorities be considered first when not all SLOs can be met with the resources available.

3.3 Global RM Configuration Space

In this section, we will describe the typical set of per-tenant configuration parameters supported by modern RMs. The *global configuration*, represented throughout this paper as the vector $\boldsymbol{x}$, used by the RM at any point is the collection of these per-tenant configurations across all tenants. $\boldsymbol{x}$ characterizes how the resource are allocated in order to process w. As we will describe in later sections, Tempo constant goal is to maximize SLO compliance adaptively for a given w by computing the optimal $\boldsymbol{x}$.

CPU, Memory, and other resources are allocated to execute the tasks in w. The resources allocated to any tenant can be captured in a fine-grained manner based on the start time, finish time, and the resource allocation vector d for each of the tasks run on behalf of the tenant.

In this paper, for ease of exposition, we will consider a uni-dimensional representation of d as an integer number of *containers* (or *slots*) as done in RMs like Mesos and YARN. Namely, a task is run in a container that is allocated on behalf of a tenant who submits the task. No two tasks can share the same container. The RM of a multi-tenant database system has a fixed total number of containers that it can allocate across all tenants at any point of time. This allocation is governed by a set of per-tenant configuration parameters falling into three categories, described next.

Resource Shares: The resource share for a tenant specifies the proportion of total resources that this tenant should get with respect to other tenants. For example, suppose there are three tenants A, B, and C with shares in the ratio 1:2:3 respectively. Suppose the database system has 12 containers that it can allocate at any point of time. Then, if all tenants have tasks to run, then tenants A, B, and C will get 2, 4, and 6 containers respectively.

Suppose a tenant does not have tasks to run in its full quota of resources. Then, the unused quota of resources will be allocated to other tenants who have tasks to run. This allocation will be proportional to the resource shares of the other tenants. In the example above, suppose tenant C has no tasks to run, but A and B have many tasks to run. Then, tenants A and B will get 4 and 8 containers respectively.

Resource Limits: For any tenant, minimum and maximum limits can be specified for the resources that this tenant can get at any point of time. In the example above where tenants A, B, and C have shares in the ratio 1:2:3 respectively, suppose all tenants have many tasks to run, but the maximum resource limit for tenant C is set to 3. Then, tenants A, B, and C will get 3, 6, and 3 containers respectively. Limits are specified to ensure two things: (i) no tenant can monopolize all resources, and (ii) critical workloads from a tenant can start running as quickly as possible.

Resource Preemption: For any tenant, a configuration can be set to preempt — after a certain time interval called a *preemption timeout* — tasks from other tenants that using resources that are rightly owed to this tenant. Such preemption will free up resources for this tenant. There are two levels of preemption timeouts. One level of preemption is when the tenant's current resource allocation is

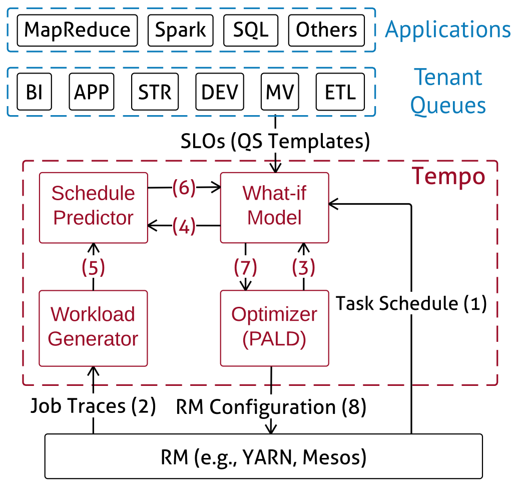

Figure 2: Tempo architecture.

below its configured resource share. The other, and more critical level, is when the tenant's current resource allocation is below its configured minimum resource limit.

Preemption is important in multi-tenant systems. Without preemption, a low-priority tenant who submitted tasks earlier than a high-priority tenant can cause the high-priority tenant to miss deadlines. Preemption can be implemented by suspending tasks or by killing tasks running in the container. While task suspension is the preferred mechanism, it is not supported in most multi-tenant systems that are commonly used today. As indicated in Section 2.3, if the two levels of preemption timeouts are not configured carefully, then preemption by killing tasks can cause a lot of wasted work and low resource utilization.

3.4 Role of Tempo

Our interviews revealed that DBAs manually tune the per-tenant RM configuration parameters in order to meet tenant SLOs. For example, at Company ABC, the RM configuration is tuned whenever tenants complain about deadline or job response time SLOs not being met. This process is brittle because it is hard for the DBAs to take into account the workload patterns and evolution, constant addition of new workloads, and the multiple objectives and tradeoffs involved. The goal of Tempo is to make this process easy and principled.

4. TEMPO

As discussed in Section 1, Tempo is designed to bring three properties to existing RMs: simplicity, self-tuning, and robustness. As part of simplicity, Tempo introduces the concept of *QS (Quantitative SLO)*. A QS is a quantitative metric defined per SLO to measure the satisfaction of the SLO at any point of time. In Section 5, we will show how the QS concept supports several tenant SLOs that arise in real-life use cases.

Operationally, the QS for an SLO can be thought of in two ways (recall Section 3.3):

1. As a function $f(\boldsymbol{x}, w)$, where $\boldsymbol{x}$ and w are described in Section 3.
2. As a function of the actual task resource allocation schedule (henceforth called *task schedule*) that is produced when w runs under $\boldsymbol{x}$.

As we will show in Section 5, it is conceptually easier for humans to understand and use the QS concept when defined in terms of the task schedule. At the same time, Tempo needs the $f(\boldsymbol{x}, w)$ notion in order to create a modular architecture that provides self-tuning and robustness. Figure 2 shows how this modular architecture drives the repeated execution of Tempo's control loop.

The Tempo control loop consists of the eight steps denoted (1)-(8) in Figure 2. The inputs to the Tempo control loop are the SLOs defined for each tenant (which can be specified conveniently via predefined templates as discussed in Section 5). Step (1) of the control loop extracts the recent task schedule for evaluating QS metrics for the input SLOs under the current RM configuration $\boldsymbol{x}$. Through Steps (2)-(8), Tempo replaces the current RM configuration $\boldsymbol{x}$ with a new one $\boldsymbol{x}'$; concluding one iteration of the control loop. Once the QS metrics for the input SLOs under $\boldsymbol{x}'$ are observed at Step (1) of the next iteration, the Tempo control loop will revert the RM configuration $\boldsymbol{x}'$ back to $\boldsymbol{x}$ if the currently observed QS metrics do not dominate the previously observed ones. This mechanism adds robustness in Tempo by guarding against performance degradation during the self-tuning approach.

Steps (2)-(8) are orchestrated by Tempo's *Optimizer* which applies a self-tuning algorithm called *PALD*. PALD is a novel multi-objective optimization algorithm that we developed for the noisy environments seen in production multi-tenant parallel database systems. As we will show in Section 6, PALD *provably* converges to a RM configuration that provides a Pareto-optimal setting for the QS metrics of the input SLOs. In addition, whenever available resources are insufficient to fully satisfy all SLOs, PALD handles the SLO tradeoffs gracefully by minimizing the largest regret across all SLO satisfactions as measured by the QS metrics.

In Steps (2)-(8), the Optimizer explores a set of RM configurations by proposing the RM configurations (3)-(4), getting the simulated task schedule (6) of the workloads (5) based on the job traces (2). The predicted QS metrics under these RM configurations are passed back to the Optimizer (7) to compute a Pareto-improving RM configuration (8). To implement these steps, the Optimizer uses three other components as shown in Figure 2: *Workload Generator*, *Schedule Predictor*, and *What-if Model*.

The Workload Generator replays historical job traces or synthesizes workloads with given characteristics. The Schedule Predictor produces the simulated task schedule of the generated workloads under given RM configurations. The What-if Model estimates the QS metrics for the input SLOs using the simulated task schedule. Together, the three components enable the Optimizer to explore the impact of different RM configurations on the input SLOs and use the PALD algorithm (described in Section 6) to produce Pareto-optimal RM configurations for these SLOs.

While proposing RM configurations in Step (3), the Optimizer meticulously generates configurations only within a given maximum distance to the currently used RM configuration. Tempo uses normalized l^2-norm as the distance metric, and allows the DBA to specify the maximum distance based on her risk tolerance. This technique further reduces the risk of causing dramatic impact on the running workloads when applying a new RM configuration; which is particularly desirable in production environments.

5. QS: QUANTIFIABLE METRICS TO MEASURE SLO SATISFACTION

A key design goal in Tempo was to provide a quantitative understanding of how the workload and RM configuration impact each SLO. We developed the QS metric which can be used to compare the relative SLO satisfactions under different workloads and RM configurations. Minimizing the QS metric improves the corresponding SLO.

The QS metric for an SLO is defined as a function of the resulting task schedule for a workload under a given RM configuration. Recall from Section 3.3 that a task schedule consists of start time, finish time, and the resource allocation d for each of the tasks run on behalf of a tenant. For ease of exposition, d can be considered as an integer number of containers as done in RMs like Mesos and YARN.

5.1 QS Metrics for Popular SLOs

We will now describe QS metrics for the common classes of SLOs that we came across in our interview (recall Section 3.2). Note that SLOs and corresponding QS metrics can be defined at different levels such as at the level of a recurring job, at the level of the entire workload of a tenant, at the level of the entire cluster, etc. In this section, we will define QS metrics at the job level, but the ideas generalize. Consider a certain interval of time L. Let J_i denote the set of jobs from tenant i which was submitted and completed during this interval. Let T_i be the set of tasks associated with J_i. Based on this notation, we can define the following QS metrics for the common SLOs.

Low average job response time: The QS metric for job response time SLO takes the average across all jobs executed by the tenant, as given by (1) where t_j^s and t_j^f are the submission and finish time of job j, respectively. $|J_i|$ represents the cardinality of the job set J_i.

$$\mathrm{QS}_{\mathrm{AJR}}\left(J_i\right) = \frac{1}{|J_i|}\sum_{j\in J_i}\left(t_j^f - t_j^s\right). \tag{1}$$

Deadlines: The QS metric for deadline SLO can be defined as the fraction of jobs of a tenant that missed their deadline. Let t_j^d be the deadline of the job j, the deadline QS metric can be defined as

$$\mathrm{QS}_{\mathrm{DL}}\left(J_i\right) = \frac{1}{|J_i|}\sum_{j\in J_i}\mathbb{I}\left(t_j^f > \gamma\left(t_j^f - t_j^l\right) + t_j^d\right), \tag{2}$$

where $\mathbb{I}(\cdot)$ is the indicator function, and γ is a slack (tolerance) when identifying the deadline violation. That is, a job j is considered violating the deadline t_j^d only if its completion is later than the deadline by a factor γ in terms of the job duration $t_j^f - t_j^l$. The slack makes the QS metric less sensitive to system variability.

High resource utilization: The resource utilization can be calculated as the integral of the fraction of overall resources allocated to the tenant over the time interval. We can use the *dominant resource usage* when multiple resource types are considered [20, 19, 44]. Note that the dominant resource usage is represented by a ratio between zero and one. When there is only a single resource type, we normalize the resource usage. we can define the QS metric for achieving high resource utilization as

$$\mathrm{QS}_{\mathrm{UTIL}}\left(J_i\right) = -\frac{1}{L}\sum_{j\in T_i} d_j\left(t_t^f - t_t^l\right), \tag{3}$$

where L be the length of the interval, and d_j is the amount of resources allocated to task j. This QS metric can also be applied to evaluate the impact of preemption by comparing the QS values computed using all tasks versus using only tasks that were not preempted.

High job throughput: The job throughput is defined as the number of jobs submitted and completed within the interval. The QS metric for achieving high job throughput is thus given by

$$\mathrm{QS}_{\mathrm{THR}}\left(J_i\right) = -\left|J_i\right|. \tag{4}$$

Resource fairness: The fairness can be defined by comparing the relative ratio of resource utilization used by the tenants versus the desired ratio. This definition is also known as the long-term fairness [45]. Let c_i denote the desired share of resources, the fairness QS metric follows

$$\mathrm{QS}_{\mathrm{FAIR}}\left(J_i\right) = -\left|c_i + \mathrm{QS}_{\mathrm{UTIL}}\left(J_i\right)\right|.$$

Furthermore, other cost models can also be used as QS metrics. For example, Personalized Service Level Agreements (PLSAs) [38] can be used as the QS metric for SQL queries.

5.2 QS Templates

To simplify the use of Tempo, we have implemented *QS templates* to enable tenants to specify SLOs declaratively. A QS template specifies: (a) a unique queue to which the tenant submits its workload, (b) a predefined QS metric for the SLO to be optimized (we currently support the QS metrics given above, but Tempo is extensible) , (c) optional constraints on one or more predefined QS metrics (e.g., a threshold on average job response time for the tenant's workload), and (d) an optional priority value (priorities are incorporated by multiplying the QS metric with the priority value). As an example, the ETL tenant may specify the following SLOs:

OPTIMIZE QS_{THR};
CONSTRAINT $QS_{DL} < 10\%$ *AND* $QS_{UTIL} < -25\%$;

In this example, the ETL tenant desires less than 10% deadline violations as well as more than 25% average resource utilization. When both constraints are satisfied, the ETL tenant favors high job throughput. To enable tenants to pick the numbers to specify in the QS templates, Tempo's control loop constantly shows the current values achieved for all QS metrics.

6. THEORETICAL FOUNDATIONS

6.1 Multi-objective QS Optimization Problem

Tempo's Optimizer solves the following vector optimization problem:

$$\begin{aligned}\arg\min_{\boldsymbol{x}\in\mathscr{X}} \quad & \left(\mathbb{E}\left[f_1\left(\boldsymbol{x},w,\xi\right)\right],\cdots,\mathbb{E}\left[f_k\left(\boldsymbol{x},w,\xi\right)\right]\right) \\ \text{s.t.} \quad & \mathbb{E}\left[f_i\left(\boldsymbol{x},w,\xi\right)\right] \le r_i \quad \forall i = 1,2,\cdots,k.\end{aligned} \tag{SP1}$$

- $f_i\left(\boldsymbol{x},w,\xi\right)$ denotes the noisy QS for the i-th tenant. ξ represents the noise and is an unknown random variable. Recall from Section 3.3 that $\boldsymbol{x}$ is the single global configuration vector used by the RM at any point of time, and $\mathscr{X}$ is the RM configuration space. Tempo has to account for the fact that the components in $\boldsymbol{x}$ corresponding to one particular tenant can affect other tenants' SLOs. For example, the priority, relative resource shares, and preemption timeout settings.
- In practice, the actual SLOs $f_i\left(\boldsymbol{x},w\right)$ cannot be computed directly, but only estimated via noisy observations or via models trained from noisy observations. For this reason, we introduce ξ, and use the What-if Model in Tempo to estimate $f_i\left(\boldsymbol{x},w,\xi\right)$. We make the standard assumption from Stochastic Approximation[30] that $\min_{x\in\mathscr{X}} f_i\left(\boldsymbol{x},w\right) = \min_{x\in\mathscr{X}}\mathbb{E}\left[f_i\left(\boldsymbol{x},w,\xi\right)\right]$for each SLO $f_i\left(\boldsymbol{x},w\right)$.
- w is a fixed workload given to Tempo, and a constant in (SP1).
- The expectation in (SP1) is with respect to ξ which is impacted by factors such as killed tasks, failed jobs, node restarts, etc.
- The r_i are the QS values corresponding to the desired SLOs. For example, the r_i for the ETL tenant in Section 5.2 are 0.1 and -0.25, respectively.

- The vector minimization in (SP1) is in a Pareto-optimal sense: an RM configuration $\boldsymbol{x}$ *dominates* another $\boldsymbol{x\prime}$ if $\mathbb{E}[f_i(\boldsymbol{x},w,\xi)] \leq \mathbb{E}[f_i(\boldsymbol{x\prime},w,\xi)]$ for $i = 1,2,\cdots,k$ and with at least one inequality. A solution $\boldsymbol{x}^\star$ cannot be dominated by any other configurations that satisfy the constraints.
- One can also prioritize certain SLOs over others in (SP1) by weighting the corresponding QS functions. For instance, to promote the priority of an SLO whose QS is $f_i(\boldsymbol{x},w,\xi)$, we can replace the QS with $\alpha f_i(\boldsymbol{x},w,\xi)$, where $\alpha > 1$ is the magnitude of the promotion.

6.2 Goals and Notation

We now present a novel *PAreto Local Descent (PALD)* algorithm for solving the multi-objective QS optimization problem (SP1). A comparison of PALD to state-of-the-art is illustrated in Table 2, in which the efficiency is based on both sample and computational complexity. In the following sections, we describe PALD and prove its properties.

Table 2: Multi-objective optimization algorithms.

	Efficient	Noisy QSs	Constraints	Pareto-optimal
SCALAR[8], MGDA[16], PESMO[23]	✓			✓
PAL[52], SMSego[41]	✓	✓		✓
ParEGO[31], EHI[17], SUR[40]				✓
MSPD[35]	✓	✓	✓	
PALD	✓	✓	✓	✓

We denote vectors and matrices by boldface symbols. The simplified notations f_i and $f_i(\boldsymbol{x})$ are used interchangeably to refer to the QS metric function $f_i(\boldsymbol{x},w,\xi)$, and we use $\boldsymbol{f}(\boldsymbol{x})$ to refer to the vector of QS functions. For each QS metric, we denote the average of N measures by $f_i(\boldsymbol{x})$.

The goal of PALD is to find a weak Pareto-optimal solution to (SP1). If a feasible solution exists, then the resulting RM configuration satisfies the "hard" SLOs represented by the constraints in (SP1), while improving the "best-effort" SLOs. If there is no feasible solution, then the resulting RM configuration balances the SLOs represented by the constraints based on max-min fairness. This feature supports prioritizing the SLOs by weighting the corresponding constraints.

6.3 Proxy Model

The key technique used in PALD is a proxy model, which transforms the original problem (SP1) to a proxy problem (SP2) such that all solutions to the proxy problem are solutions to the original one, but not the other way around. We show that the proxy problem can be solved efficiently.

First, it should be noted that the well-known *weighted sum scalarization* ([8])—which converts the multi-dimensional QS vector to a scalar by taking a weighted sum of the QS functions—does not apply in this case; for it does not ensure the first set of constraints in the problem (SP1). For example, consider two RM configurations and two QS functions. Suppose that the QS vectors corresponding to the two solutions are $(5,5)^\top$ and $(0,7)^\top$, respectively. Let $\boldsymbol{r} = (6,6)^\top$. When the weights are equal, the optimization using weighted sum scalarization yields the QS vector $(0,7)^\top$, which does not dominate $\boldsymbol{r} = (6,6)^\top$.

Our solution PALD solves the following proxy problem:

$$\begin{aligned} \arg\min_{\boldsymbol{x}\in\mathscr{X}} \quad & \boldsymbol{c}^\top [\boldsymbol{f}(\boldsymbol{x}) - \rho \max(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{r})] \\ \text{s.t.} \quad & \mathbb{E}[f_i(\boldsymbol{x})] \leq r_i \quad \forall i = 1,2,\cdots,k. \end{aligned} \tag{SP2}$$

Here, $\boldsymbol{c}$, which is a positive vector, and $\rho < 1$ are two parameters whose values will be described in Section 6.3.1. The parameter ρ penalizes those QS functions $f_i(\boldsymbol{x}) > r_i$, and is independent of the vector $\boldsymbol{c}$. This is an advantage over *conic scalarization* [28]. One special case is that when $\rho = 0$, the problem (SP2) becomes the weighted sum scalarization.

THEOREM 1. *For any arbitrary positive vector* $\boldsymbol{c}$ *and parameter* $\rho < 1$*, every solution of* (SP2) *is a solution for* (SP1).

PROOF. Let $s_i(\boldsymbol{x}) = c_i[f_i(\boldsymbol{x}) - \rho\max(f_i(\boldsymbol{x}),r_i)]$, the objective of problem (SP2) can be written as

$$\begin{aligned} \sum_i s_i(\boldsymbol{x}) &= \sum_{i:f_i(\boldsymbol{x})\leq r_i} s_i(\boldsymbol{x}) + \sum_{j:f_j(\boldsymbol{x})>r_j} s_j(\boldsymbol{x}) \\ &= \sum_{i:f_i(\boldsymbol{x})\leq r_i} c_i[f_i(\boldsymbol{x}) - \rho r_i] + \qquad (5) \\ &\quad \sum_{j:f_j(\boldsymbol{x})>r_j} c_j(1-\rho) f_j(\boldsymbol{x}). \qquad (6) \end{aligned}$$

Both (5) and (6) are strictly monotonically increasing with respect to $f_i(\boldsymbol{x})$, so is the objective (SP2). Consider a solution $\boldsymbol{x}$ for (SP2). Suppose that $\boldsymbol{x}$ is not a weak Pareto-optimal solution for (SP1), then there exists another weak Pareto-optimal solution $\boldsymbol{x\prime}$ for the problem (SP1) that dominates $\boldsymbol{x}$. However, this contradicts the hypothesis that $\boldsymbol{x}$ is a solution for the problem (SP2), due to the monotonicity. □

6.3.1 Parameters

We now derive the parameters $\boldsymbol{c}$ and ρ in the proxy model (SP2). PALD uses Stochastic Approximation [30] for solving the proxy problem, in which the gradients are estimated using the well-known LOESS [13]. Let $s(\boldsymbol{x})$ denote the objective of the proxy problem (SP2), the update for each iteration is given by

$$\boldsymbol{x}^{new} = \boldsymbol{x}^{old} - \alpha \boldsymbol{\nabla}_{\boldsymbol{x}} s(\boldsymbol{x}), \tag{SGD}$$

where α is the step size. The parameters $\boldsymbol{c}$ and ρ are chosen such that the above update does not increase those QS functions $f_i(\boldsymbol{x}) \geq r_i$. We thereby obtain $\forall i: f_i(\boldsymbol{x}) \geq r_i$, $-\alpha \boldsymbol{\nabla}_{\boldsymbol{x}}{}^\top f_i(\boldsymbol{x}) \boldsymbol{\nabla}_{\boldsymbol{x}} s(\boldsymbol{x}) \leq 0$, or equivalently $\boldsymbol{\nabla}_{\boldsymbol{x}}{}^\top f_i(\boldsymbol{x}) \boldsymbol{\nabla}_{\boldsymbol{x}} s(\boldsymbol{x}) \geq 0$. The parameters are also chosen to best improve those violated constraints $f_i(\boldsymbol{x}) \geq r_i$. Fixing $\boldsymbol{c}$, ρ is obtained by solving

$$\begin{aligned} \arg\max_{\rho} \min_{i:f_i(\boldsymbol{x})\geq r_i} \quad & \boldsymbol{\nabla}_{\boldsymbol{x}}{}^\top f_i(\boldsymbol{x}) \boldsymbol{\nabla}_{\boldsymbol{x}} s(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{\nabla}_{\boldsymbol{x}}{}^\top f_i \boldsymbol{\nabla}_{\boldsymbol{x}} s(\boldsymbol{x}) \geq 0, \quad \forall i: f_i(\boldsymbol{x}) \geq r_i \\ & \boldsymbol{c} \geq 0, \quad \rho < 1. \end{aligned} \tag{RHO}$$

Note that the objective of the proxy model (SP2) is not differentiable at points $\{\boldsymbol{x} \in \mathscr{X} : \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{r}\}$, and we need to condition on the subgradients. Let us first assume that $\partial s(\boldsymbol{x})/\partial f_j(\boldsymbol{x})\big|_{x=r} = c_j(1-\rho)$. The objective of the problem (RHO) can be rewritten as

$$\sum_j c_j \boldsymbol{\nabla}_{\boldsymbol{x}}{}^\top f_i \boldsymbol{\nabla}_{\boldsymbol{x}} f_j - \rho \sum_{j:f_j(\boldsymbol{x})\geq r_j} c_j \boldsymbol{\nabla}_{\boldsymbol{x}}{}^\top f_i \boldsymbol{\nabla}_{\boldsymbol{x}} f_j. \tag{7}$$

Based on the range of the subgradient of $s(\boldsymbol{x})$, we can bound ρ. To satisfy the first set of constraints in the problem (RHO) at an indifferentiable point, we have that

$$\min_{i,\frac{\partial s}{\partial f_j}:f_i(\boldsymbol{x})\geq r_i} \sum_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j \frac{\partial s}{\partial f_j} \geq 0. \tag{8}$$

Now consider separately two cases $\rho \geq 0$ and $\rho < 0$. When $\rho \geq 0$ the inequality (8) is equivalent to $\forall i : \boldsymbol{\nabla_x} f_i \neq \boldsymbol{0} \wedge f_i(\boldsymbol{x}) \geq r_i$ that

$$\begin{aligned}(1-\rho)\, c_i \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_i &\geq -\sum_{j:j\neq i} \min_{\partial s/\partial f_j} \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j \frac{\partial s}{\partial f_j} \\ &= -(1-\rho) \sum_{\substack{j:j\neq i,\\ \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j \geq 0}} c_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j \\ &\quad - \sum_{\substack{j:j\neq i,\\ \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j < 0}} c_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j,\end{aligned}$$

which simplifies to

$$0 \leq \rho \leq \min_{\substack{i:\boldsymbol{\nabla_x} f_i \neq \boldsymbol{0},\\ f_i(\boldsymbol{x})\geq r_i}} \frac{\sum_j c_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j}{\sum_{j:\boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j \geq 0} c_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j}.$$

Similarly, we can obtain the bound for the case $\rho < 0$. It should be noted that these bounds are useful only when the following conditions are satisfied:

$$\sum_j c_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j \geq 0, \quad \forall i : \boldsymbol{\nabla_x} f_j \neq \boldsymbol{0} \wedge f_i(\boldsymbol{x}) \geq r_i. \tag{9}$$

These conditions can be satisfied for convex QS functions, using the vector $\boldsymbol{c}$ described in MGDA [16]. Combining the results arrives at the optimal choice of ρ for the problem (RHO):

$$\rho_* = \begin{cases} \min\limits_{\substack{i:\boldsymbol{\nabla_x} f_j \neq \boldsymbol{0},\\ f_i(\boldsymbol{x})\geq r_i}} \dfrac{\sum_j c_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j}{\sum_{j:\boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j \geq 0} c_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j}, & \rho \geq 0 \\ \max\limits_{\substack{i:\boldsymbol{\nabla_x} f_j \neq \boldsymbol{0},\\ f_i(\boldsymbol{x})\geq r_i}} \dfrac{\sum_j c_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j}{\sum_{j:\boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j < 0} c_j \boldsymbol{\nabla_x}^\top f_i \boldsymbol{\nabla_x} f_j}, & \rho < 0. \end{cases}$$

The sign of the parameter ρ depends on the last term of the objective (7) as to maximize the objective. To deliver the above optimal ρ_*, the vector $\boldsymbol{c}$ must also satisfy the conditions (9).

To achieve max-min fairness of SLOs, PALD chooses $\boldsymbol{c}$ that improves the most violated constraint, through the following linear program.

$$\begin{aligned} \text{maximize} \quad & z \\ \text{subject to} \quad & \boldsymbol{J}_{i:f_i(\boldsymbol{x})\geq r_i} \boldsymbol{J}^\top \boldsymbol{c} \geq z\boldsymbol{1} \\ & \boldsymbol{c} \geq 0, \quad z \leq \varepsilon. \end{aligned}$$

Here, $\boldsymbol{J}$ is the Jacobian of the QS vector, and $\boldsymbol{J}_{i:f_i(\boldsymbol{x})\geq r_i}$ denotes the rows of the Jacobian $\boldsymbol{J}$ indexed by $i : f_i(\boldsymbol{x}) \geq r_i$. ε is an arbitrary positive constant, and the solution vector $\boldsymbol{c}$ is normalized using any desirable metrics such as the l^2-norm. The first set of constraints correspond to the QS functions $f_i(\boldsymbol{x}) \geq r_i$, and these are the only QS functions that need to be convex in PALD. Thus, PALD provides better support for non-convex QS optimization as compared to MGDA. Moreover, randomly choosing different initial points can also help deal with non-convex QS functions in this sense.

7. WHAT-IF MODEL

Tempo's Optimizer depends on the What-if Model to estimate the values of noisy QS metrics $f_i(\boldsymbol{x}, w, \xi)$. The What-if Model breaks each prediction into two steps and leverages the Workload Generator and Schedule Predictor respectively for these steps. Recall that the QS metric is expressed as a function of the task schedule of w under $\boldsymbol{x}$. The Workload Generator is responsible for generating the workload, and the Schedule Predictor is responsible for generating the task schedule given the workload and the RM configuration.

7.1 Workload Generation

As discussed in Section 3.1, there are two ways to generate w in Tempo: sampling from historical traces or using a statistical model of the workload. Tempo offer users both options in the Workload Model(see Figure2). As a rule of thumb, using a job trace yields more realistic w, and is thereby preferred whenever traces are available. In contrast, the statistical model, which is usually trained from historical traces, has some key advantages. The model can be used to generate multiple synthetic w's with perturbed distributions in order to test the sensitivity of parameter settings. More importantly, the model can be used to generate w with extended characteristics such as a growth in data size by 30%. For example, we developed a statistical model based on one month of historical traces from Company ABC's production database workload. The workload distributions from Company ABC (reported further in the evaluation section) are similar to the distributions described in [42]. In particular, the task duration approximately follows a lognormal distribution, and the job arrival approximately follows a Poisson process.

7.2 Fast Schedule Prediction

Given a workload generated as above, the Schedule Predictor in Figure 2 estimates the task schedule of the workload under a given RM configuration. Since the What-if Model needs to explore the impact of many different RM configurations, fast prediction of schedules can speed up Tempo's optimization process significantly.

For very fast task schedule simulation, we implemented a Schedule Predictor for the RMs used in Hadoop, Spark, and YARN using time warp mechanism [27]. Our implementation computes the cluster resource usage at only the submission time, tentative finish time, and possible preemption time of each task, based on the workload information and RM configuration parameter settings. This technique helps the Predictor get rid of actually running the tasks as well as synchronization within the RM.

To extend to other RMs, Tempo can leverage existing RM simulators that have already been developed for several popular systems, such as Borg [49], Apollo [11], Omega [43], MapReduce [48, 24, 22], and YARN [7]. Most of these existing simulators are designed to reproduce the real-time behavior of the RM, which is a superset of our goal of computing the task schedule efficiently.

8. EVALUATION

We now report an end-to-end evaluation of Tempo using production workload traces from Facebook, multiple customers of Cloudera [12], as well as Company ABC. We apply Tempo to four real-life scenarios and show, respectively, the improvements in job response time, resource utilization, adaptivity to workload variations, and predictive resource provisioning. In the experiments where a baseline performance is needed for comparison, we used resource allocations as determined by expert DBAs and cluster operators in Company ABC. The following insights emerge from the evaluation:

- Tempo can tailor the resource allocation to SLO-driven business-critical workloads, and offers tenants the freedom to specify SLOs.

- Tempo improves the resource utilization by 15%, and job response time for best-effort tenants by 50% under 25% slack without breaking the deadlines for production workloads.
- Tempo effectively adapts to workload variations by periodically updating the RM configuration using a recent workload window.
- Tempo can help DBAs and cluster operators determine the appropriate cluster size for their multi-tenant parallel database for the given SLOs and workloads, minimizing the overall resource costs.

These results are due to: 1) informed resource allocation which takes into account the workload characteristics revealed from historical job traces; and 2) optimized RM configurations aiming for the SLOs because Tempo makes the connection between the RM configuration and SLOs more transparent and predictable.

8.1 Validating the schedule prediction

We begin by validating the task schedule prediction on a 700-node production cluster at Company ABC. In particular, we measure the accuracy of the prediction using one week's production workload from six independent tenants, as described in Table 1. The workload consist of approximately 60,000 jobs and 35 million production tasks collected in a noisy environment where there were job and task failures, jobs killed by users and DBAs, and node blacklisting, failures, and restarts. Figure 3 shows the key statistics of the workload.

The schedule prediction for the 35 million tasks from six tenants takes just 4 minutes, or approximately 150,000 tasks per second. We compare the predicted task schedule and the observed schedule based on the traces, and compute the prediction error. Both the relative absolute error (RAE) and the relative squared error (RSE) are used as the error metrics. The RAE and RSE of tenant i are defined respectively as

$$\mathrm{RAE}_i = \frac{\sum_j |p_{ij} - l_{ij}|}{\sum_j |l_{ij} - \mathbb{E}_j[l_{ij}]|}, \quad \mathrm{RSE}_i = \sqrt{\frac{\sum_j (p_{ij} - l_{ij})^2}{\sum_j (l_{ij} - \mathbb{E}_j[l_{ij}])^2}} \ .$$

Here p_{ij} and l_{ij} represent the predicted and observed finish time of job j for tenant i, respectively. Table 8.1 gives the RAE and RSE for the estimated job finish time. As can be seen, the highest error (24.4%) was incurred for the MV tenant in Company ABC. Most jobs from MV were long-running jobs, especially with large duration of reduce tasks. We observed a considerable amount of killed reduce tasks for MV due to preemptions. For killed and failed tasks, the task start time and finish time are not recorded accurately in workload traces; which explains why MV has a higher prediction error than others.

Table 3: Job finish time estimation errors for each tenant.

Tenant	RAE	RSE	Tenant	RAE	RSE
BI	0.1585	0.2210	STR	0.1610	0.1463
DEV	0.2195	0.2267	MV	0.2318	0.2437
APP	0.1812	0.1599	ETL	0.1210	0.1908

8.2 End-to-end evaluation

The end-to-end experiments involve four real-life scenarios, and were performed on a 20-node Amazon EC2 cluster with m3.xlarge instances. The production workload traces from Company ABC, Facebook, and Cloudera customers were scaled and replayed on the EC2 cluster using SWIM [12]. In addition, the initial RM configuration was derived directly from the expert one created by DBAs for Company ABC's production database. Each end-to-end experiment involves approximately 30,000 tasks from two tenants, and each Tempo control loop explores 5 RM configuration candidates. Thus, one Tempo control loop requires prediction for 150,000 tasks, which takes one second.

8.2.1 Mix of deadline-driven and best-effort workloads

The first scenario involves two tenants running workloads which come with the deadline SLO specified with $\mathrm{QS}_{\mathrm{DL}}$, and the low average job response (AJR) time SLO specified with $\mathrm{QS}_{\mathrm{AJR}}$, respectively. The experiment aimed to obtain an RM configuration which is better than the expert one used in production. In particular, under the new RM configuration, *every* job from the deadline-driven workload must complete no later than the completion of the same job under the expert RM configuration. This is a strict constraint, where the deadlines in $\mathrm{QS}_{\mathrm{DL}}$ are given by the completion times of deadline-driven jobs under the expert RM configuration, and the corresponding $r_i = 0$ (for 0% deadline violations). Another constraint involving $\mathrm{QS}_{\mathrm{AJR}}$ enforces that the average job response time of the best-effort workloads under the new RM configuration cannot be greater than the average job response time (r_i) under the expert configuration.

When counting the number of deadline violations, a 25% slack, i.e., $\gamma = 0.25$, is used in $\mathrm{QS}_{\mathrm{DL}}$ to reduce the sensitivity to noise, since even the workloads under the same RM configuration with a slack 0 ($\gamma = 0$ in $\mathrm{QS}_{\mathrm{DL}}$) can yield a large deadline violation fraction (up to 83%).

Figure 4 shows the SLOs (QS values) at each iteration in the Tempo control loop. At the iteration 0, the initial expert RM configuration was used. The RM configuration was then iteratively optimized by the Tempo control loop. It can be seen that, at convergence, the improvements in average job response time of the best-effort tenant are 50% and 58% for 25% and 50% slack, respectively. The gap between the improvements is relatively small, i.e., 8%. One reason is that both improvements benefited from the reduced contention for resources, which is confirmed in the next experiment. In addition, the fraction of deadline violations first drops and then breaks even at convergence. This trend is due to the fact that once the Pareto frontier is reached, we cannot improve one SLO without sacrificing another.

8.2.2 Improving resource utilization

In addition to the previous scenario, this experiment considered a third SLO, high resource utilization, which is specified with $\mathrm{QS}_{\mathrm{UTIL}}$. We focused exclusively on MapReduce workloads due to the observation of significant task preemptions in production. The experiment added two constraints corresponding to the map container utilization and reduce container utilization, respectively. The r_i's were set according to the measured map and reduce container utilization under the expert RM configuration. The results show fewer preemptions under the Tempo optimized RM configuration as well as improvements in job response time subject to the deadline SLOs.

As we discussed, preemption happens when a tenant has been starved for a certain period of time (the configured preemption timeout), killing a certain number of most recently launched tasks from other tenants. Thus, preemption results in lost work and decreased resource utilization. Each tenant can specify a per-tenant preemption timeout in the RM configuration, and these settings are difficult to get right manually (even for experts) due to their complex connections to workloads and SLOs.

We observed a significant number of preempted MapReduce tasks on the production cluster at Company ABC. Figure 5 shows the

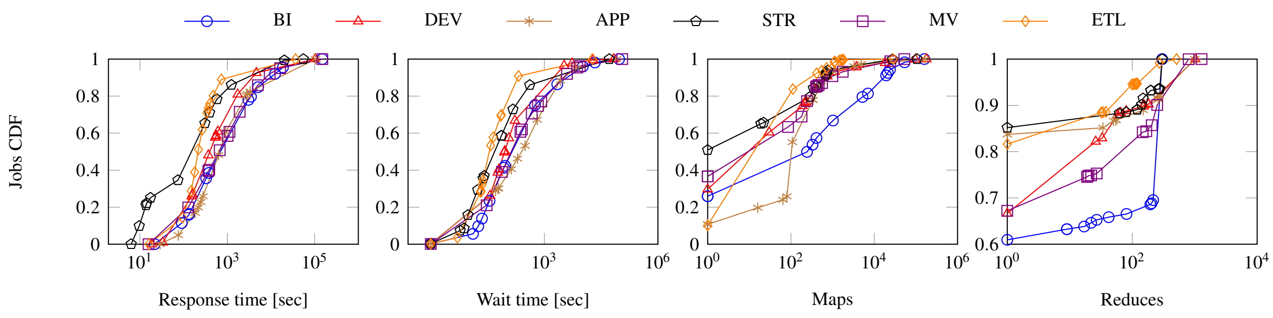

Figure 3: Key statistics of Company ABC's workloads.

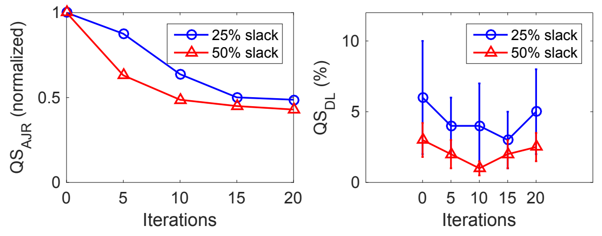

Figure 4: Average job response time for the best-effort tenant (left) and fraction of deadline violations for the deadline-driven tenant (right) at each iteration.

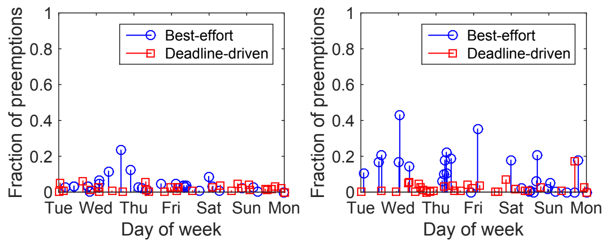

Figure 5: Task preemptions for MapReduce workloads at Company ABC. On the left shows the preempted map tasks, and the preempted reduce tasks are given on the right.

map and reduce preemptions over the period of one week. During this period, 6% map tasks and 23% reduce tasks had been preempted, and the reduce preemptions were mostly from the best-effort tenant. The main reason was that the workloads of the best-effort tenant contain mostly long-running reduce tasks, as shown in Figure 6.

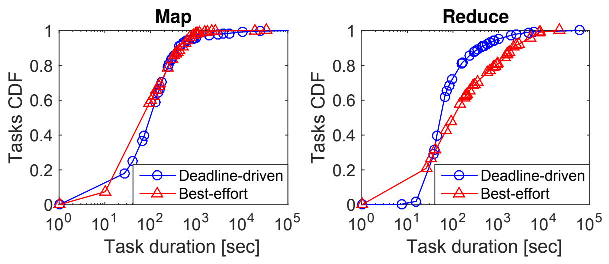

Figure 6: Task duration distributions for MapReduce workloads at Company ABC.

Figure 7 shows the SLOs under the original expert RM configuration and the Tempo optimized RM configuration. As can be seen, the optimized resource allocation delivers 22% improvement in the average job response time of the best-effort tenant workloads and 10% in the deadline QSs. Another improvement is in the utilization of reduce containers, while the utilization of map containers remains at the same level. The results are consistent with our observations of preemption statistics, and the improvements in reduce container utilization is due to the alleviated preemptions. In particular, the preemption timeout settings in the Tempo-optimized RM configuration had been self-tuned appropriately to the workload distribution.

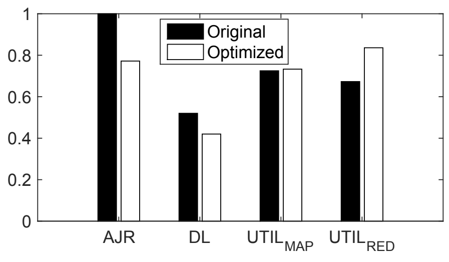

Figure 7: SLOs under the original and optimized (slack = 0) RM configuration: AJR, DL, $UTIL_{MAP}$, and $UTIL_{RED}$ are the average job response time of the best-effort tenant, fraction of deadline violations for the deadline-driven tenant, map container utilization, and reduce container utilization, respectively.

8.2.3 Performance of PALD Vs. State-of-the-art

In this experiment, we compare the proposed PALD algorithm to state-of-the-art for multi-objective QS optimization (see Table 2). Specifically, we consider six tenants A to F with average job response time, deadline, and throughput SLOs, and evaluate the configurations produced by the algorithms in comparison. These tenants run SWIM-generated workloads based on production traces from Cloudera clients and Facebook.

Since measuring QSs and making QS estimations are the most time-consuming phases in Tempo, we naturally rule out algorithms with large sample complexity, e.g., evolutionary algorithms. The QS optimization (SP1) also requires the solution to support constraints.

This condition removes PAL, PESMO, and SUR from consideration. MSPD is the state-of-art which supports constraints; however, MSPD does not seek a Pareto-optimal solution. The goal of MSPD is instead to optimize one selected objective while satisfying all the other objectives with respect to the specified thresholds.

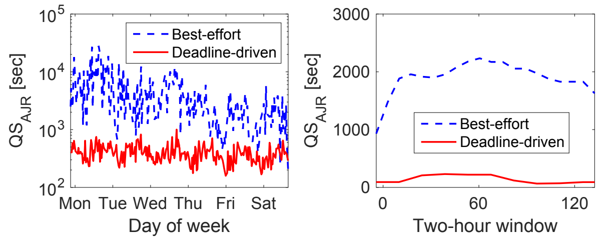

Figure 9: Instant job response time distributions. On the left shows the production workloads of Company ABC over the period of a week. On the right gives the two-hour experiment workloads on EC2 using Facebook and Cloudera traces.

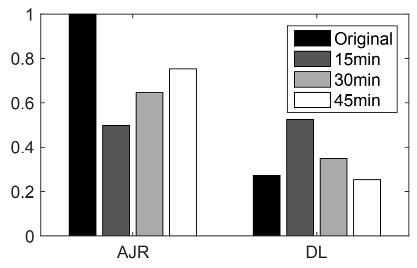

Figure 10: SLOs for different interval lengths in Tempo control loop. AJR denotes the normalized average job response time of the best-effort workloads, and DL represents the fraction of deadline violations (computed via QS_{DL} with slack $\gamma = 25\%$).

For the above reasons, we compare three algorithms: SCALAR, MSPD, and PALD. SCALAR is the widely-used scalarization which serves as a naive approach. The experiment starts with the same arbitrary configuration, and performs the optimization using different algorithms. The same number of iterations are performed until every algorithm has converged. Each iteration takes 1 min for every algorithm and uses two hours of job history for computing SLOs. Figure 8 shows the normalized results, as compared to the expert configuration, for the initial arbitrary configuration and configurations optimized by different algorithms. In particular, the SLOs achieved under the expert configuration are specified as constraints for these algorithms to try to force the optimized configuration to dominate the existing one.

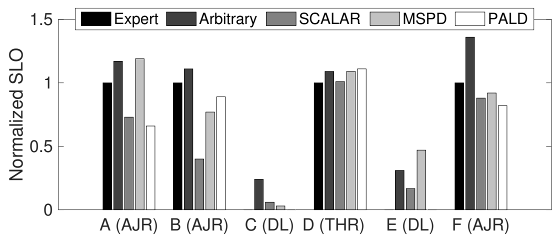

Figure 8: Comparison of algorithms for QS optimization. Expert is the baseline configuration and others are normalized accordingly. AJR, DL, THR are respectively the average job response time, deadline, and throughput(#jobs/hr) SLOs.

As can be seen, the arbitrary configuration resulted in decreased performance for all tenants except D. Unsurprisingly, SCALAR leads to increased deadline violations for C and E while achieving decent improvements in AJR SLOs due to the ignorance of SLO constraints. Since MSPD does not seek Pareto-improving solutions, the resulting configuration does not dominate the expert one. We also noted that MSPD assumes a convex configuration space, which generally does not hold for RMs. For example, the preemption parameters are in binary and integer domains. The configuration optimized by the proposed PALD is Pareto-improving as desired.

8.2.4 *Adaptivity to workload variations*

In this experiment, we applied Tempo to meet SLOs under slowly changing workload distributions. Figure 9 depicts the instant job response time distribution for deadline-driven and best-effort tenants. The instant job response time is computed using the moving average of a 30-minute window. As can be seen, the instant job response time of deadline-driven workloads exhibits a periodic pattern while the job response time of the best-effort workloads changes dramatically over time.

Recall that each iteration of the Tempo control loop uses a fixed-length interval of most recent job traces as input. The next experiment evaluates how different interval lengths impact Tempo's performance.

Figure 10 shows the SLOs under the original expert RM configuration and Tempo-optimized RM configurations for interval length 15min, 30min, and 45min. Similarly, the experiment uses SLOs specified with QS_{AJR}, and QS_{DL} (25% slack). As can be seen, a small window size favors the average job response time of the best-effort workloads while leading to a higher percent of deadline violations. According to the results, the 45min interval length yields a similar fraction of deadline violations as the original RM configuration, but a 22% improvement in the average job response time of the best-effort workloads. The results show that Tempo can adapt to workload variations using a small interval length.

8.2.5 *Resource provisioning and cutting costs*

The last experiment demonstrates the application of Tempo to resource provisioning, estimating the minimum amount of resources needed to meet the given SLOs. This application can help users do better resource planning and cut overprovisioning costs. In addition, this application can bridge the gap in resource allocation between the development cluster and the production cluster, that is, converting the resource allocation on the development cluster for use in the production cluster.

The experiment involves running the same given deadline-driven workloads and best-effort workloads on three EC2 clusters with 20 nodes (100%), 10 nodes (50%), and 5 nodes (25%), respectively. Tempo was used to estimate the SLOs of the workloads when executed on the 100% cluster, using traces respectively from the 100% cluster, 50% cluster, and 25% cluster. This experiment mimics the scenario in which users collect traces of the workload on the current cluster, and would like to know how a new cluster size will impact the SLOs. (From our experience, this use case is common at companies like LinkedIn and Yahoo.) In this case, Tempo can serve as a key component in the decision-making for resource provisioning.

Figure 11 gives the SLO estimation errors using traces from equal and smaller clusters. As can be seen, Tempo can predict—with the error no more than 20%—the SLOs of the current workloads run on a double-size cluster; using traces collected from the current cluster. Predicting the SLOs of the current workloads run on a quadruple-size cluster results in a maximum error of 35%.

9. RELATED WORK

General-purpose RMs. Most Resource Managers (RMs) that are deployed on multi-tenant "big data" database systems today like

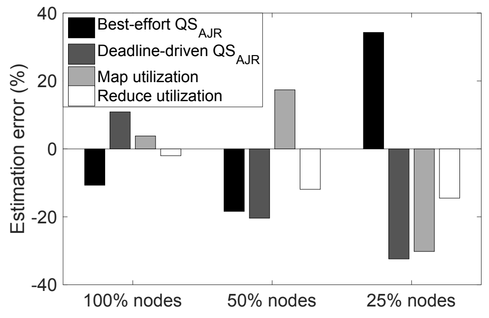

Figure 11: Errors in SLO estimation using traces based on equal and smaller cluster sizes.

RedShift, Teradata, Vertica, Hadoop, and Spark are based on resource allocation principles such as static resource partitioning [1], *max-min fairness* [4, 3], *dominant resource fairness* [20, 19, 44], or reservations [14]. Variants also consider data locality [26], job placement constraints [20], and multi-resource packing of tasks to machines [21]. Moreover, RMs (usually called Workload Managers) in parallel database systems like IBM DB2 PE, RedShift, Teradata, and Vertica allow DBAs to specify rules to dynamically adjust the resource allocation of tenants, as well as define user-defined events relevant to workload management and actions to be taken based on them. RedShift and Vertica use resource pools where each pool has parameters such as resource limits, priorities, and maximum concurrency like the RM configuration described in Section 3.3.

A large body of recent research focuses on developing general-purpose RMs like YARN[46] and Mesos[25]. The main efforts lie in scalability, responsiveness, and fault-tolerance of the RMs. Omega [43] proposes parallelism, shared state, and lock-free optimistic concurrency control for increased scalability. Sparrow [39] leverages load-balancing techniques to make the scheduler more responsive for scheduling low-latency tasks. Fuxi [51] enhances the fault tolerance and scalability of RMs by introducing transparent failure recovery features and a failure detection mechanism. Apollo [11] takes into account the data locality and server load to achieve high-quality scheduling decisions. Apollo also introduces correction mechanisms to cope with unexpected cluster dynamics, sub-optimal estimations, and other abnormal runtime behaviors.

The above RMs generally provide effective control and isolation over resources. However, they have limited support for application-level and tenant-oriented SLOs, and often rely on DBAs to guess the right capacities.

Single SLO-driven RMs. Many existing RMs aim to achieve a single type of SLO. In [29], the authors develop a deadline estimation model and apply real-time scheduling to meet job deadlines. ARIA [47] provides support for job deadlines by profiling jobs and modeling resource requirements in order to complete before the deadline. WOHA [33] improves workflow deadline satisfactions in Hadoop. Amoeba [9] brings lightweight elasticity to compute clusters by splitting original tasks into smaller ones, and allowing safe exit of a running task and later resuming the task by spawning a new task for its remaining work. Pisces [44] delivers datacenter-wide per-tenant performance isolation and fairness for multi-tenant cloud storage.

Tenant-oriented RMs. A handful of RMs have been proposed to provide tenant performance isolation. Unlike resource isolation, these RMs typically incorporate capacity estimation to meet tenant-oriented SLOs. Pulsar [10] uses *virtual datacenter abstraction (VDC)*, which is similar to the QS abstraction in Tempo, to describe the SLOs for a tenant. Unlike Tempo, Pulsar focuses on scenarios where resources are sufficient, and does not guarantee Pareto optimality of SLOs. The effectiveness of Pulsar also relies on the accuracy of user-specified cost functions in VDCs as well as resource demand estimation. Thus, Pulsar can be sensitive to noise in both cost and demand estimation. Retro [34] supports resource-limited scenarios and delivers max-min fairness across SLOs by balancing the progress of applications (referred to as workflows). Retro does not guarantee the Pareto-optimality of tenant SLOs. For example, one may use techniques like multi-resource packing [21] to obtain Pareto-improving configurations with similar relative fairness ratios among tenants.

10. CONCLUSION

Providing end-to-end tenant performance isolation while achieving high resource utilization in multi-tenant "big data" database systems is an important problem. The vast majority of resource managers deployed on multi-tenant database systems today rely on DBAs to continually configure low-level resource settings to support tenant performance isolation. This process is brittle and increasingly hard as workloads evolve, data and cluster sizes change, and new workloads are added. In this paper, we presented a framework, Tempo, which enables DBAs to work with high-level SLOs conveniently. In particular, we showed that Tempo has provable robustness properties which enforce tenant performance isolation, and more desirably Pareto-optimal SLOs. The evaluation reports that Tempo is self-tuning and robust for achieving guaranteed SLOs in production database systems.

11. REFERENCES

[1] Amazon redshift. `http://goo.gl/nS8cQH`.
[2] Facebook corona. `https://goo.gl/MN9VpK`.
[3] Hadoop capacity scheduler. `https://goo.gl/hh6Ood`.
[4] Hadoop fair scheduler. `https://goo.gl/8Ov2nj`.
[5] Teradata. `http://goo.gl/TjU7qR`.
[6] Vertica. `http://tinyurl.com/ovgufev`.
[7] Yarn scheduler load simulator. `https://goo.gl/JNUPoj`.
[8] F. B. Abdelaziz. Solution approaches for the multiobjective stochastic programming. 216(1):1–16, 2012.
[9] G. Ananthanarayanan, C. Douglas, R. Ramakrishnan, S. Rao, and I. Stoica. True elasticity in multi-tenant data-intensive compute clusters. In *SOCC*, pages 24:1–24:7, 2012.
[10] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, and E. Thereska. End-to-end performance isolation through virtual datacenters. In *OSDI*, pages 233–248, 2014.
[11] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *OSDI*, pages 285–300, 10 2014.
[12] Y. Chen, S. Alspaugh, and R. Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. 5(12):1802–1813, 08 2012.
[13] W. S. Cleveland and S. J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. 83:596–610, 1988.
[14] C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao. Reservation-based scheduling: If you're late don't blame us! In *SOCC*, pages 2:1–2:14, 2014.

[15] S. Das, V. Narasayya, F. Li, and M. Syamala. Cpu sharing techniques for performance isolation in multi-tenant relational database-as-a-service. In *PVLDB*, 2013.

[16] J.-A. Désidéri. Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. Tome 350(Fascicule 5-6):313–318, 03 2012.

[17] A. Emmerich. The computation of the expected improvement in dominated hypervolume of Pareto front approximations, 2008.

[18] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In *EuroSys*, pages 99–112, 2012.

[19] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, pages 323–336, 2011.

[20] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Choosy: Max-min fair sharing for datacenter jobs with constraints. In *EuroSys*, pages 365–378, 2013.

[21] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. 44(4):455–466, 08 2014.

[22] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu. MRSim: A discrete event based MapReduce simulator. 6:2993–2997, 08 2010.

[23] D. Hernández-Lobato, J. M. Hernández-Lobato, A. Shah, and R. P. Adams. Predictive entropy search for multi-objective bayesian optimization. In *NIPS Workshop on Black-box Learning and Inference*, 2015.

[24] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, pages 261–272, 2011.

[25] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, pages 295–308, 2011.

[26] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *SOSP*, pages 261–276, 2009.

[27] D. Jefferson, H. Sowizral, and R. Corporation. *Fast Concurrent Simulation Using the Time Warp Mechanism: Part I, Local Control*. Fast Concurrent Simulation Using the Time Warp Mechanism: Part I, Local Control. Rand Corporation, 1982.

[28] R. Kasimbeyli. A conic scalarization method in multi-objective optimization. 56(2):279–297, 2013.

[29] K. Kc and K. Anyanwu. Scheduling hadoop jobs to meet deadlines. In *CLOUDCOM*, pages 388–392, 2010.

[30] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. 23:462–466, 1952.

[31] J. Knowles. Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. 10(1):50–66, 09 2006.

[32] W. Lang, S. Shankar, J. Patel, and A. Kalhan. Towards multi-tenant performance slos. In *ICDE*, 2012.

[33] S. Li, S. Hu, S. Wang, L. Su, T. Abdelzaher, I. Gupta, and R. Pace. Woha: Deadline-aware map-reduce workflow scheduling framework over hadoop clusters. In *ICDCS*, pages 93–103, 2014.

[34] J. Mace, P. Bodik, R. Fonseca, and M. Musuvathi. Retro: Targeted resource management in multi-tenant distributed systems. In *NSDI*, pages 589–603, 05 2015.

[35] M. Mahdavi, T. Yang, and R. Jin. Stochastic convex optimization with multiple objectives. In *NIPS*, pages 1115–1123. 2013.

[36] V. Narasayya, S. Das, M. Syamala, B. Chandramouli, and S. Chaudhuri. Sqlvm: Performance isolation in multi-tenant relational database-as-a-service. In *CIDR*, January 2013.

[37] V. Narasayya, S. Das, M. Syamala, S. Chaudhuri, F. Li, and H. Park. A demonstration of sqlvm: Performance isolation in multi-tenant relational database-as-a-service. In *SIGMOD*, pages 1077–1080, 2013.

[38] J. Ortiz, V. T. de Almeida, and M. Balazinska. Changing the face of database cloud services with personalized service level agreements. In *CIDR*, 2015.

[39] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *SOSP*, pages 69–84, 2013.

[40] V. Picheny. Multiobjective optimization using gaussian process emulators via stepwise uncertainty reduction. 25(6):1265–1280, 2014.

[41] W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze. Multiobjective optimization on a limited budget of evaluations using model-assisted $\mathcal{S}$-metric selection. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature: PPSN X*, pages 784–794, 2008.

[42] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou. Workload characterization on a production hadoop cluster: A case study on taobao. In *IEEE International Symposium on Workload Characterization*, pages 3–13, 2012.

[43] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *EuroSys*, pages 351–364, 2013.

[44] D. Shue, M. J. Freedman, and A. Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *OSDI*, pages 349–362, 2012.

[45] S. Tang, B.-s. Lee, B. He, and H. Liu. Long-term resource fairness: Towards economic fairness on pay-as-you-use computing systems. In *ICS*, pages 251–260, 2014.

[46] 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 *SOCC*, pages 5:1–5:16, 2013.

[47] A. Verma, L. Cherkasova, and R. H. Campbell. Aria: Automatic resource inference and allocation for mapreduce environments. In *ICAC*, pages 235–244, 2011.

[48] A. Verma, L. Cherkasova, and R. H. Campbell. Play it again, simmr! In *CLUSTER*, pages 253–261, 2011.

[49] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *EuroSys*, 2015.

[50] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigumus. Intelligent management of virtualized resources for database systems in cloud environment. In *ICDE*, pages 87–98, 2011.

[51] 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. 7(13):1393–1404, 08 2014.

[52] M. Zuluaga, A. Krause, G. Sergent, and M. Püschel. Active learning for multi-criterion optimization. In *ICML*, 2013.