Semantics-Aware Prediction for Analytic Queries in MapReduce Environment

Weikuan Yu
Florida State University
yuw@cs.fsu.edu

Zhuo Liu
Auburn University
zhuoliu@auburn.edu

Xiaoning Ding
New Jersey Inst. of Technology
xiaoning.ding@njit.edu

ABSTRACT

MapReduce has emerged as a powerful data processing engine that supports large-scale complex analytics applications, most of which are written in declarative query languages such as HiveQL and Pig Latin. Analytic queries are typically compiled into execution plans in the form of directed acyclic graphs (DAGs) of MapReduce jobs. Jobs in the DAGs are dispatched to the MapReduce processing engine as soon as their dependencies are satisfied. MapReduce adopts a job-level scheduling policy to strive for balanced distribution of tasks and effective utilization of resources. However, there is a lack of query-level semantics in the purely task-based scheduling algorithms, resulting in resource thrashing among queries and an overall degradation of performance. Therefore, we introduce a semantic-aware query prediction framework to address these problems systematically. Our framework includes three major techniques: cross-layer semantics percolation, selectivity estimation, and multivariate time prediction for analytic queries. Multivariate query prediction allows us not only to gauge the dynamic size of analytics datasets, but also to accurately predict the resource usage (e.g., numbers of map and reduce tasks) of individual MapReduce jobs and whole queries. In addition, the accurate prediction and queuing of queries can be potentially exploited by Hadoop scheduling for optimizing overall query performance. Based on the query prediction, our case study scheduler demonstrates significant performance improvement compared to traditional Hadoop schedulers.

CCS CONCEPTS

• **Software and its engineering → Software performance**; **Runtime environments**;

KEYWORDS

MapReduce, Semantics-Aware, Analytics Query, Scheduling

ACM Reference Format:
Weikuan Yu, Zhuo Liu, and Xiaoning Ding. 2018. Semantics-Aware Prediction for Analytic Queries in MapReduce Environment. In *ICPP '18 Comp: 47th International Conference on Parallel Processing Companion, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, Article 3, 9 pages. https://doi.org/10.1145/3229710.3229713

ICPP '18 Comp, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6523-9/18/08…$15.00
https://doi.org/10.1145/3229710.3229713

1 INTRODUCTION

According to IDC [12], the total digital data generated per year would reach 40,000 exabytes by 2020. Among such a gigantic amount of data, 33% can bring valuable information if analyzed. However, currently only 0.5% of total data has been analyzed due to limited analytic capabilities. MapReduce [7] and its open-source implementation Hadoop [1] have become powerful engines for processing *Big Data* and extracting precious knowledge for various business and scientific applications. Analytics applications often impose diverse yet conflicting requirements on the performance of underlying MapReduce systems; such as high throughput, low latency, and fairness among jobs. For example, to support latency-sensitive applications from advertisements and real-time event log analysis, MapReduce must provide fast turnaround time.

Because of their declarative nature and ease of development, analytics applications are often created using high-level query languages. These analytic queries are transformed by compilers into an execution plan of multiple MapReduce jobs, which are often depicted as direct acyclic graphs (DAGs). A job in a DAG can only be submitted to MapReduce when its dependencies are satisfied. A DAG query completes when its last job is finished. Thus, the execution of analytic queries is centered around dependencies among jobs in a DAG, as well as the completion of jobs along the critical path of the DAG. On the other hand, to support MapReduce jobs from various sources, the lower level MapReduce systems usually adopt a two-phase scheme that allocates computation, communication, and I/O resources to two types of constituent tasks (map and reduce tasks) from concurrently active jobs. For example, the Hadoop Fair Scheduler (HFS) and Hadoop Capacity Scheduler (HCS) strive to allocate resources among map and reduce tasks to aim for good fairness among different jobs and high throughput of outstanding jobs. When a job finishes, a scheduler will immediately select tasks from another job for resource allocation and execution. However, these two jobs may belong to DAGs of two different queries. Such interleaved execution of jobs can cause prolonged execution for different queries and delay the completion of all queries. Besides, the lack of query semantics and job relationships in these schedulers can also cause unfairness to queries of distinct structures, e.g., chained or tree-shaped queries.

Such problems are due to the mismatch between system and application objectives. While the schedulers in the MapReduce processing engine focus on job-level fairness and throughput, analytic applications are mainly concerned with query-level performance objectives. This mismatch of objectives often leads to prolonged execution of user queries, resulting in poor user satisfaction. As Hive [28] and Pig Latin [23] have been used pervasively in data warehouses, such problems become a serious issue and must be timely addressed. More than 40% of Hadoop production jobs at

Yahoo! run as Pig programs [13]. In Facebook, 95% Hadoop jobs are generated by Hive [18].

In this paper, we propose a semantic-aware query prediction framework that can address these problems systematically. In this framework, three techniques are introduced including cross-layer semantics percolation, selectivity estimation, and multivariate time prediction. First, cross-layer semantics percolation allows the flow of query semantics and job dependencies in the DAG to the MapReduce scheduler. Second, with rich semantics information, we model the changing size of analytics data through selectivity estimation, and then build a multivariate model that can accurately predict the execution time and resource usage of different individual jobs and queries. Furthermore, based on our multivariate time prediction, query-based scheduling can be introduced to optimize overall query serving performance.

Our contributions from this research are listed as follows.

- We introduce cross-layer semantics percolation to bridge the semantic gap between the MapReduce schedulers and declarative analytic queries.
- We create a multivariate query prediction model that can leverage query semantics to accurately predict the data dynamics along the query DAG, as well as the execution time for jobs and queries.
- We have demonstrated the effectiveness and accuracy of our framework and its potential benefits of improving query throughput and response performance.

The rest of this paper is organized as follows: In Section 2, we present our proposed modeling framework. In Section 3, we describe our semantics-aware selectivity estimation of various query operators. We then elaborate our multivariate prediction model in Section 4. Section 5 presents the performance evaluation of our model. Finally, we provide a review of related research on execution modeling of MapReduce jobs in Section 6, and then conclude the paper in Section 7.

2 SEMANTICS-AWARE QUERY PREDICTION

In the current MapReduce-based query processing framework, a physical execution plan for Hive queries is generated as a DAG of MapReduce jobs after being processed by the parser and semantic analyzer. These DAGs are then submitted to Hadoop according to precedence constraints. Traditional Hadoop schedulers such as HCS and HFS are adopted to allocate resources to runnable MapReduce jobs, which may belong to different DAGs (meaning that the jobs stem from different queries). As we have mentioned in Section 1, the lack of query semantics awareness can result in inefficient execution of analytic queries, which can be seen in both HCS and HFS. In the former, it is possible for jobs belonging to different queries to have their executions interleaved. In the latter, resource allocation can be divided too thinly among jobs. Furthermore, the lack of query semantics and job relationship knowledge within schedulers can also cause unfairness to different types of queries. The remainder of this section investigates and proposes a solution for the impact of semantics-unawareness on query scheduling.

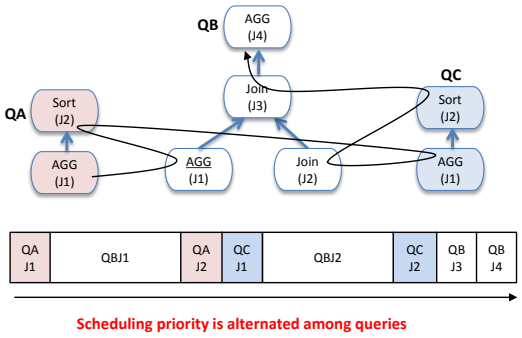

Figure 1: Under HCS, scheduling priority is alternated among different queries due to semantic unawareness, causing resource thrashing.

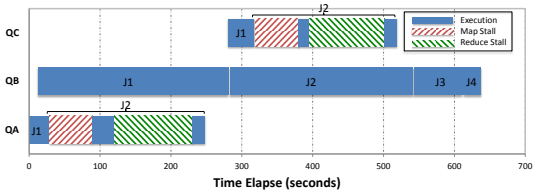

Figure 2: QB and QC's completions are delayed by 3× because of resource thrashing.

2.1 Motivation

TPC-H [3] represents a popular online analytical workload. We have conducted a test using a mixture of three TPC-H queries which contains two instances of Q14 and one instance of Q17. Q14 evaluates the market response to a production promotion in one month. Q17 determines the loss of average yearly revenue if some orders are not properly fulfilled in time. For convenience, we denote the two instances of Q14 as QA and QC, and the only instance of Q17 as QB. Figure 1 shows the DAGs for these three queries and their constituent jobs. Both QA and QC have 10 GB of input data and are composed of two jobs: Aggregate (AGG) and Sort. QB is a query composed of four jobs and with larger input data size of 100 GB.

In our experiment, we submit QA, QB, and QC one after another to our Hadoop cluster. Figure 1 shows that the scheduling priority in this test is alternated among three queries' jobs under the HCS scheduler, which causes serious resource thrashing among the different queries. Figure 2 further demonstrates the performance of the HCS scheduler. Since J1 and J2 from QB arrive before QA-J2 and QC-J2 respectively, they are scheduled for execution. As a result, QA-J2 and QC-J2 are blocked from getting container resources. We can see that QA-J2 and QC-J2 both experience execution stalls due to the lack of available containers. Such stalls delay the execution of QA and QC (small queries) by 3× more than when they are running alone.

When a job is satisfied in HCS, the scheduler immediately selects tasks from another job to receive resource allocation and begin execution. It is possible for these two jobs to exist in the DAGs of

different queries. Such interleaved execution of jobs can increase the makespan of different queries, which results in a significant delay of the completion for all queries.

The HFS scheduler is known for its monopolizing behavior, causing different jobs to stalls [27, 30]. Applying the same experimental setup as the HCS experiment described above to HFS, we have observed similar execution stalls of QA and QC. For succinctness, the results are not shown here. Suffice to say that, in HFS, the thinly divided resource allocation among all active jobs can result in suboptimal performance of queries.

Therefore, due to the lack of knowledge about query compositions and actual resource demand, Hadoop schedulers cause execution stalls and performance degradation of small queries. In a large-scale analytics system with many concurrent query requests, the issue of execution stalls caused by semantic-oblivious scheduling is exacerbated.

2.2 Solution

For all jobs in a DAG, Hive submits one job to the JobListener when the job's dependency has been satisfied. Jobs from various queries are linearly ordered on the Hadoop scheduler, which then assigns containers according to either the HCS or HFS policy.

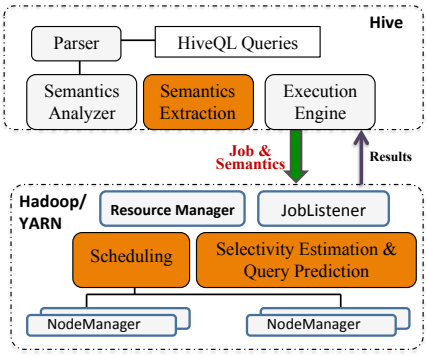

Figure 3: Semantic-Aware Query Prediction

Clearly, all the query-level semantics are lost when Hadoop receives a job from Hive. In addition, the traditional Hadoop scheduler can only see the presence of individual jobs, not their parent queries. As a result, there is no coordination among jobs to ensure the best overall progress for the query. To address these issues systematically, we propose a *semantics-aware query prediction framework*.

Our framework (shown in Figure 3), extracts the query semantics information during the query compilation and execution plan during construction phases. The semantics information is then submitted alongside the query's jobs to the scheduler, including the DAG of jobs, the dependencies among jobs, the operators and predicates of the query, and the associated input tables. Selectivity estimation uses the semantic information to estimate the size of data along a query's DAG. It reflects the selectivity ratio between input data and intermediate results, and the number of input and output tuples.

Based on the estimation of selectivities among different types of operators, we further introduce the multivariate time prediction model, including the prediction for query execution time and resource usage. The knowledge of query execution time and resource usage is critical for efficient scheduling. A query usually consists of multiple MapReduce jobs, each of which have their separate input tables scans, materialization phase, and De/serialization phases. Therefore, predictions of individual jobs coupled with the consideration of global query semantics can provide insight about job execution statistics and dynamic behaviors. In addition, there are multiple phases of data processing and movement during the execution of a job, leading to the dynamically changing data size inside a job and along the DAG of a query. A good model needs to reflect these data dynamics [9]. We first describe the selectivity estimation of different jobs in a query in Section 3, and then elaborate the integration of selectivity estimation in the execution time prediction model in Section 4.

3 SEMANTICS-AWARE SELECTIVITY ESTIMATION

As shown in Figure 4, for a MapReduce job in a query DAG, the output of its map tasks provides the input for its reduce tasks. One job's output is often taken as a part of the input of its succeeding job in the same DAG. The input size of a job directly affects its resource usage (number of map and reduce tasks) during execution in MapReduce. In addition, the data size of map output (intermediate data) affects the execution time of reduce tasks and that of the downstream jobs in the query. Accurate modeling of execution for the job and/or the query requires a good estimation of the dynamic data size during execution. We divide this requirement into two metrics: *Intermediate Selectivity* and *Final Selectivity*. Let us denote the input of a job as D_{In}, the intermediate data as D_{Med}, and the output as D_{Out}. The Intermediate Selectivity (IS) is defined as the ratio between D_{Med} and D_{In}, and the Final Selectivity (FS) as the ratio between D_{Out} and D_{In}.

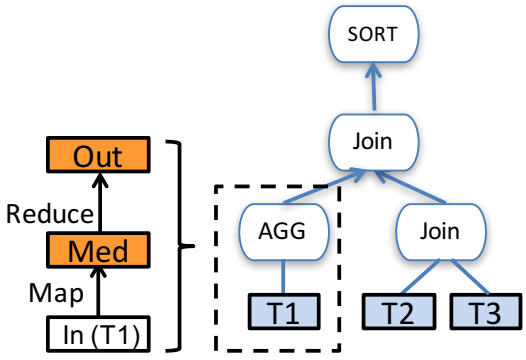

Figure 4: Intermediate Selectivity and Final Selectivity

3.1 Selectivity Estimation

On Hive, users can choose a variety of different operators to express their requests (queries). When queries are compiled into DAGs, the query operators are grouped into either *major operators* or *minor operators*. Global shuffle/aggregation operators (such as groupby, orderby, and join) are major operators, placing all other operators (such as normal range, equality predicates, and map-side join) in the

category of minor operators. We first categorize MapReduce jobs in Hive into three types with respect to the major operators: Groupby, Join, and Extract (which includes orderby and all remaining major operators). We then introduce two abstractions of this estimation, *intermediate selectivity* and *final selectivity*, and apply them to our Groupby, Join, and Extract categories.

3.1.1 Intermediate Selectivity. In general, a job's intermediate selectivity is determined by the semantics of its predicate and projection clauses on the input tables (i.e., the selectivity of these clauses; the rows and columns returned by a query).

The selectivity of a projection clause, S_{proj} can be calculated as the ratio between the average width of the attributes selected by the projection clause and the average size of tuples in a table. As such, we can see that the estimation of S_{proj} relies on statistical information of the table.

Let $|In|$ and $|Med|$ denote the numbers of tuples in the input table and the intermediate data, respectively. If there are no local combine operations in the Map tasks, the selectivity of a predicate clause, S_{pred}, can be calculated as the fraction of the number of tuples in the intermediate data and the number of tuples in the input table; $S_{pred} = |Med|/|In|$. We consider the effects of local combines when discussing the Groupby category. Off-line histograms are built for the attributes of the input table to estimate S_{pred}. Assuming piece-wise uniform distribution of attribute values, equi-width histograms [24] are built on tables' attributes to be filtered through a MapReduce job and stored on HDFS.

Extract: An extract operation usually scans one input table and we have its intermediate selectivity as $IS = S_{pred} \times S_{proj}$.

Groupby: In a Groupby operation, a local combine can further reduce the intermediate data. Let S_{comb} denote the combination selectivity. We can calculate Groupby intermediate selectivity as

$$IS = S_{comb} \times S_{proj}. \tag{1}$$

The calculation of S_{comb} itself is dependent on the distribution of the groupby keys, which we will describe below.

Suppose a job performs a groupby operation on Table T's keys x and y. Let $T.d_{xy}$ denote the product from the numbers of distinct keys for x and y, and $|T|$ as the number of tuples in table T. If the groupby keys are all clustered in the table, S_{comb} is calculated as

$$S_{comb} = min(1, \frac{T.d_{xy}}{|T| \times S_{pred}}) \times S_{pred} = min(S_{pred}, \frac{T.d_{xy}}{|T|}). \tag{2}$$

Otherwise, if the groupby keys are randomly distributed, S_{comb} is then calculated as $S_{comb} = min(S_{pred}, \frac{T.d_{xy}}{|T|/N_{maps}})$, where N_{maps} denotes the number of map tasks in the job. We omit the calculation for other minor cases.

Join: A join operation may select tuples from two or more input tables. The intermediate selectivity of a join depends on the predicate selectivity, projection selectivity, and percentage of each table's data in the input of the job. We describe the calculation of IS for a simple join job of two tables. Let r_1 represent the percentage of one table's data in the total input of a job, $r_2 = (1 - r_1)$ for the other table. We can calculate IS for a join job with two input tables as:

$$IS = S_{pred\,1} \times S_{proj\,1} \times r_1 + S_{pred\,2} \times S_{proj\,2} \times (1 - r_1). \tag{3}$$

3.1.2 Final Selectivity. Let us denote $|Out|$ as the number of tuples in the output (Out), and W_{out} as the average width of those tuples. A job's final selectivity is calculated as

$$FS = |Out| \times W_{Out}/D_{In}. \tag{4}$$

Specifically, $FS = 1$ for map-only jobs. Therefore, for each category of jobs, we must compute $|Out|$ in order to calculate FS.

Extract: In Hive, there are two common extract operations, "$limit\ k$" and "$order\ by$". For the former, $|Out| = min(|In|, k)$; and for the latter, $|Out| = |In|$.

Groupby: A groupby operation may use one or more keys. The number of tuples in the output is determined by the cardinalities of the keys and the predicates on the keys. We show the calculation for an example operation with one key. For a table T, let us denote the cardinality of Key x as $T.d_x$. A groupby operation on key x will have $|Out| = min(T.d_x, |T| \times S_{pred})$.

Join: There are many variations of join operations. Multiple operations may hierarchically form a join tree. We focus on a few common join operations of two or three tables to illustrate the calculation of $|Out|$, as well as the most important join case: an equi-join between a primary key and a foreign key (where tables should obey referential integrity).

An equi-join operation from tables T_1 and T_2 will have $|Out| = |T_1 \bowtie T_2| = |T_1| \times |T_2| \times \frac{1}{max(T_1.d_x,\ T_2.d_x)}$ if join keys follow uniform distribution [26]. However, uniform distribution is rare in practical cases. Additionally, this approach only applies for multiple joins that share a common join key. In our work, we assume a piece-wise uniform distribution, where each equi-width bucket key follows uniform distribution.

Let T_{1i} denote the i-th bucket of the equi-width histogram for a join operation on Key x. Then, we can calculate the number of tuples in a join job's result set as:

$$|T_1 \bowtie T_2| = \sum_{i=1}^{n} |T_{1i}| \times |T_{2i}| \times \frac{1}{max(T_{1i}.d_x,\ T_{2i}.d_x)} \tag{5}$$

Since $(T_{1i} \bowtie T_{2i}).d_x = min(T_{1i}.d_x,\ T_{2i}.d_x)$, the equation above can be evolved to calculate the join selectivity for shared-key joins on three or more tables.

For chained joins with unshared keys, e.g., where T_1 and T_2 join on Key x, but T_2 and T_3 join on Key y, we leverage the techniques introduced in [6] by acquiring the updated piece-wise distribution of Key y after the first join For natural joins (each operator joins one table's primary key with another table's foreign key) with local predicates on each table, selectivities are accumulated along branches of the join tree, thus the number of tuples in the result set can be approximated as:

$$\begin{aligned} &|T_1.pred_1 \bowtie T_2.pred_2 \bowtie \cdots \bowtie T_n.pred_n| \\ &\qquad = S_{pred\,1} S_{pred\,2} \ldots S_{pred\,n}\, max(|T_1|, |T_2|, \ldots, |T_n|) \end{aligned} \tag{6}$$

3.2 An Example of Selectivity Estimation

We use a modified TPC-H [3] query Q11 as an example to demonstrate the estimation of selectivities. Figure 5 shows the flow of selectivity estimation. This query is transformed into two join jobs and one groupby job. In Job 1, the predicate on the nation table has a predicate selectivity of 96% and, is relayed to the upcoming jobs

along the query tree. Thus we can predict *IS* and *FS* for Job 1 and Job 2 according to the equations above. In Job 3, since the groupby key (partkey) has a cardinality of 200,000 that is much less than input tuples of this job, the cardinality of Job 3 is approximated as 200,000.

```
SELECT ps_partkey, sum(ps_supplycost*ps_availqty)
FROM nation n JOIN supplier s ON
  s.s_nationkey=n.n_nationkey AND n.n_name<>'CHINA'
JOIN partsupp ps ON
  ps.ps_suppkey=s.s_suppkey
GROUP BY ps_partkey;
```

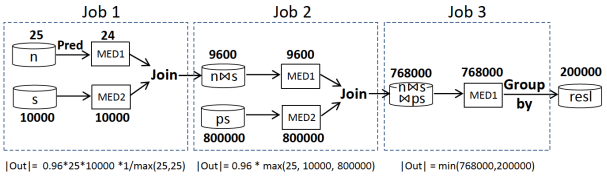

Figure 5: An Example of Selectivity Estimation

4 MULTIVARIATE TIME PREDICTION MODEL

Based on the estimation of selectivities, we build a multivariate time prediction model for jobs, tasks and queries. We focus on the three operations as we have discussed: Extract, Groupby and Join. Next we first describe our time prediction models for different types of jobs, then the task-based time models for improving prediction scalability, and finally the resource usage prediction for a whole query.

4.1 Time Prediction for Different Types of Jobs

As listed in Table 1, we rely on several input features to model the execution time. To simplify the jobs with Groupby or Extract operator, we include three parameters D_{In}, D_{Out} and D_{Med} which can provide sufficient modeling accuracy. D_{Out} and D_{Med} are estimated based on *IS* and *FS* mentioned in Section 3. When calculating D_{Med}, the types of operator should be considered. For example, Extract's $IS = S_{pred} \times S_{proj}$, so the number of tuples of intermediate result $D_{Med} = D_{In} \times IS$. The D_{Med} of Groupby and Join operations can be processed in a like manner. Besides, different types of jobs display distinct selectivity characteristics. Thus we include the operator type as part of our generalized multivariate model.

Table 1: Input Features for the Model

Name	Description
O	The Operator Type: 1 for Join, 0 for others
D_{In}	The Size of Input Data
D_{Med}	The Size of Intermediate Data
D_{Out}	The Size of Output Data
$P(1-P)D_{Med}$	The Data Growth of Join Operators

Table 1 lists the input features of our model. The three parameters D_{In}, D_{Out}, and D_{Med} provide sufficient modeling accuracy as the characteristics of operator are recognized during their calculations. However, for a join job, these parameters are not enough to reflect the growth of data sizes because the number of tuples can be the Cartesian product of input tables. Let $|T_1|$ and $|T_2|$ denote the number of tuples for the two input tables of a join operator. We define P as the ratio between the number of tuples in the larger filtered table and the sum of the tuples in the two filtered tables, i.e.,

$$P = \frac{max(|T_1|S_{pred1}, |T_2|S_{pred2})}{|T_1|S_{pred1} + |T_2|S_{pred2}}, \ 0 < P < 1. \quad (7)$$

So $P(1-P)$ reflects the factor of a join operator, $P(1-P) \in (0, \frac{1}{4}]$. To better estimate prediction accuracy, we apply P only for Join operator.

Based on these input features, we formulate a linear model with a set of coefficients $\vec{\theta} = [\theta_0, \theta_1, ..., \theta_m]$ to predict the job execution time (ET) as:

$$ET = \theta_0 + \theta_1 D_{In} + \theta_2 D_{Med} + \theta_3 D_{Out} + \theta_4 O \times P(1-P) D_{Med}, \quad (8)$$

where all inputs are explained in Table 1. To learn the model coefficients, we build up a training set using queries from TPC-H and TPC-DS benchmarks [3], and data size ranges from 1GB to 100GB. Queries in these workloads perform data analysis tasks resulting in complex query plans with many operations We collected the job execution time for 5,647 MapReduce jobs by running 1000 queries on a given Hadoop cluster.

More features may lead to better prediction accuracy [32]. However, they can cause more monitoring overhead and are difficult to obtain in real-time. Thus the rationale behind our choice of features is to balance the need of accuracy with the complexity of extracting input features. Note that in the paper we concentrate on time prediction for analytic queries, for other non-relational workloads such as User-Defined Functions (UDFs), there are some available solutions in recent work [21, 29].

4.2 Time Prediction for Map and Reduce Tasks

Even though we have a job-level multivariate linear model, the parameters' value ranges for various jobs can sometimes go far beyond our training set, thereby causing underestimation of job execution time [19]. To deal with such issue, similar to the job models introduced in Section 4.1, we further empirically build prediction models for the average execution time of a job's map or reduce tasks based on the task type, the operator type, job scale, the per-task input size and output size. The task time estimation models are used for both improving the scalability of our job time prediction and resource usage for queries. Similar to Equation 8, we model the task execution time as follows:

$$ET_i = \kappa_0 + \kappa_1 TD_{In\_i} + \kappa_2 TD_{Out\_i} + \kappa_3 O \times P(1-P) TD_{In\_i}, \quad (9)$$

where ET_i is the predicted time for the i-th task, TD_{In_i}, TD_{Out_i} and TD_{Med_i} are the data size of i-th task input and task output, respectively. κ_0, κ_1, κ_2, and κ_3 are the set of coefficient parameters for the task execution model.

When the size of input data D_{In} is known, the total number of map tasks and reduce tasks, TD_{In_i} could be easily calculated. Besides, $TD_{Out\_i} = IS \times TD_{In\_i}$ for map tasks, and $TD_{Out\_i} = FS \times TD_{In\_i}$ for reduce tasks, where *IS* and *FS* are intermediate selectivity and final selectivty, respectively. Note that *IS* and *FS* are varied among different types of operators. The last term of Equation 9 is used to account for data growth from operators such as join. To determine the coefficients, the task model has also been

trained by the queries from TPC-H and TPC-DS benchmarks to gather the execution time of MapReduce jobs on Hadoop cluster.

4.3 Scheduling with Semantics-Aware Prediction

We need to ensure all queries be fairly treated with comparable slowdown ratios so that small queries can turn around faster while big queries still get their fair share of time and resources for execution if delayed by a certain degree. However, in selecting queries that are organized as DAGs of jobs, we cannot solely rely on the temporal resource demand of a query, i.e. its remaining time that can be estimated from our multivariate model. A query and all of its jobs often employ a dynamic number of tasks during its execution. Each task may have its own execution time, CPU, memory and I/O resource demand.

We introduce a simple metric to take into account of the time and resource demand of a query or a job called **W**eighted **R**esource **D**emand (*WRD*). *WRD* is intended as an metric to estimate how much system resource will be required to complete a query or a job. A query's *WRD* is calculated as:

$$WRD = \sum_{i=1}^{n} MT_i \times N_{M\,i} + RT_i \times N_{R\,i}, \tag{10}$$

where MT_i denotes the predicted map task time, $N_{M\,i}$ denotes the remaining number of map tasks, RT_i denotes the predicted reduce task time and $N_{R\,i}$ the remaining number of reduce tasks, for an arbitrary job i of the query. For a job that is large enough to occupy all the available containers in the system, its execution time can be approximated as the job's *WRD* divided by the number of available containers plus scheduling overheads. Based on the semantic-aware query prediction framework, we further propose a *Smallest WRD first* query scheduling scheme (SWRD) as a case study for improving average query performance. To be specific, this scheduling scheme prioritizes the query with smallest weighted resource usage for optimizing overall query performance.

5 EVALUATION

In this section, we carry out extensive experiments to evaluate the effectiveness of the framework with a diverse set of analytic query workloads.

5.1 Experimental Settings

Testbed: We have implemented our semantic-aware scheduling framework in Hive v0.10.0 and Hadoop v1.2.1. Our experiments are conducted on a cluster of 9 nodes, each of which features two 2.67GHz hex-core Intel Xeon X5650 CPUs, 24GB memory and two 500 GB Western Digital SATA hard drives. According to the resource available on each node, we configure 12 containers per node. The heap size for map and reduce tasks is set as 1GB and the HDFS block size as 256 MB. All other Hadoop parameters are the same as the default configuration. We employ Hive with the default configuration, while allowing the submission of multiple jobs into Hadoop.

Benchmarks and Workloads: To validate our model, we build up a training set using queries from TPC-H and TPC-DS benchmarks [3]. The data size ranges from 1 GB to 100 GB. Our validation test uses about 1,000 queries, which are converted into 5,647 MapReduce jobs. Among them, 3/4 queries are used as the training set while the rest are used as the part of the test set. In addition, we add 150 GB to 400 GB scale queries into the test set for assessing the model's scalability.

Table 2: Composition of Bing and Facebook Workloads

Bin	Input Size	Number of Queries	
		Bing	Facebook
1	1-10 GB	44	85
2	20 GB	8	4
3	50 GB	24	8
4	100 GB	22	2
5	>100 GB	2	1

To evaluate the efficiency of our case study scheduler (SWRD), with the TPC-H and TPC-DS queries, we build two workloads based on the workload composition on Facebook and Bing production systems characterized in [5]. For brevity, we name them *Bing workload* and *Facebook workload*, respectively. Table 2 summarizes the composition of the Bing and Facebook workloads. The queries are divided into 5 bins based on their input sizes. The queries are submitted into the system following a random Poisson distribution.

5.2 Accuracy for Job Time Prediction

Table 3: Accuracy Statistics

Types	R-squared accuracy	Avg Error
Groupby	96.75%	8.63%
Join	92.71%	14.40%
Extract	84.64%	9.38%
TestSet	N/A	13.98%

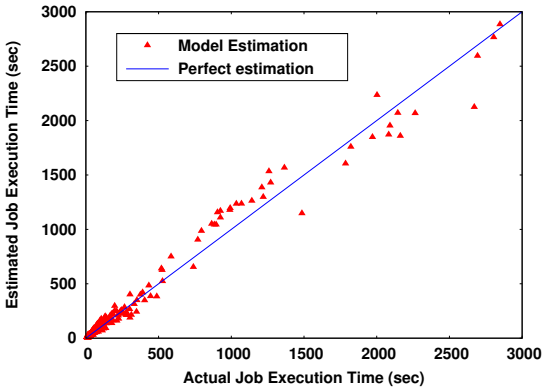

Figure 6: Accuracy of Job Execution Prediction

Table 3 summarizes the R-squared accuracy and the average error rate of our model for jobs of each operator. R-squared accuracy shows how well data fit a statistical model and a value approaching

1 indicates a good fit. The average error rate for the test set of jobs is 13.98%. The error rate indicates the deviation of predicted estimations from the perfect line. The prediction of Extract does not fit the model well compared with others. However it has less deviation, because there are more dots that are not exactly on the perfect estimation line but very close to the line. The Join operator performs an opposite way that the small number of non-fitted dots scatter a little far from the perfect line. Even though it is caused by uncertainty of the substantial sum of Join results, our precise prediction results still demonstrate the correctness of our time prediction model of different jobs.

The time prediction accuracy of our model is shown in Figure 6. The X-axis denotes the actual job execution time while the Y-axis denotes the predicted job execution time. The straight line demonstrates a perfect prediction. We can observe that our model can accurately predict the execution time of MapReduce jobs through a careful process of selectivity estimation based on a few input parameters.

5.3 Accuracy for Task Time Prediction

Table 4 and 5 summarize the R-squared accuracy and test set prediction accuracy for map tasks and reduce tasks with three different operators, respectively. In Table 4, the imprecision mostly comes from the Join operator. As the number of tuples after Join can be the Cartesian product of input tables, the scalability of the results is tough to maintain. However, the selectivity ratio for join operators as mentioned in Section 4.1 helps preserve the stability of the time prediction model.

We are able to predict most types of tasks with small error rate based on our observation, which can be utilized for improving the scalability for predicting long-running jobs. In addition, such close estimation of task execution also allows us to determine the weighted resource demands of all queries, with which we can then select the best query for execution.

Table 4: Accuracy Statistics for Map Task Time Prediction (training set)

Types	R-squared accuracy	Avg Error
Join	85.6%	16.27%
Groupby	92.4%	24.8%
Extract	92.74%	14.5%
Together	87.05%	20.5%

Table 5: Accuracy Statistics for Reduce Task Time Prediction (training set)

Types	R-squared accuracy	Avg Error
Join	85.83%	14.23%
Groupby	98.82%	4.67%
Extract	90.03%	6.18%
Together	90.68%	7.4%

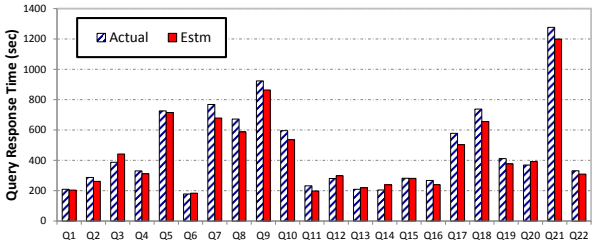

Figure 7: Accuracy of Query Response Time Prediction

5.4 Validation for Predicted Query Execution

The execution time of a query can be approximated as the sum of execution times of all jobs along the critical path of its DAG and other large jobs which are able to use up the system's resource. Thus we directly use our time prediction model of tasks to predict the execution of whole jobs, as a job is combined with several tasks.

We compare the actual execution time and estimated execution time of 100 GB TPC-H queries. As shown in Figure 7, the deviation between the estimation and actual results are very closed. The average prediction error rate can be as low as 8.3%. Again, this error rate adequately validates the accuracy of our prediction model and strengthens the correctness of selectivity estimation and alongside Join selectivity ratio in Section 4.1.

5.5 Benefits of the Semantics-Aware Scheduling

In this section, we use Bing and Facebook workloads to test the overall performance of semantics-aware scheduling. This evaluation helps us to understand how we can exploit the semantics-aware query prediction for supporting efficient scheduling under large-scale concurrent query workloads.

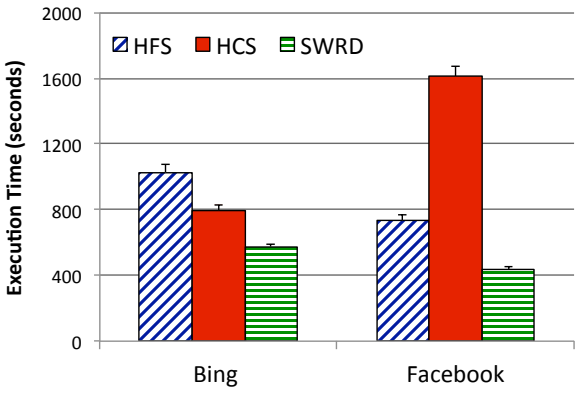

Figure 8: Average Query Response Times

Figure 8 summarizes the average query execution times of the Bing and Facebook workloads under three scheduling policies. Among all scheduling policies, SWRD achieves the best performance by leveraging the semantics information that ensures all

queries are fairly treated with negligible slowdowns. HCS has reversed performance compared to HFS, because of the characteristics of HCS that maximizes the utilization of the cluster. As the resources always have the higher priority in HCS, the semantics are broken severely under HCS than HFS. However, HFS neglects the resources utilization. Our SWRD considers the resource demands and the time, which avoids breaking the semantics and delaying priorities. Compared to HFS, SWRD reduces the average query response times by 40.2% and 43.9%, respectively. Compared to HCS, the average query response times of SWRD are decreased by 72.8% and 27.4%, respectively. This clearly demonstrates the capability of SWRD to handle query workloads consisting of queries with the help of semantics-aware query prediction.

6 RELATED WORK

In this section, we review recent work on query modeling and scheduling in the MapReduce environment.

Morton et al. [21] proposed the Paratimer model to estimate the progress of Pig queries, which are translated into DAGs of MapReduce jobs. Their model incorporated dataset cardinality and unit processing time. Verma et al. [29, 36] provided a "work/speed" time model for the relationship of execution time and available resources. Both time models require pre-execution profiling or debug runs of the same job in order to acquire the necessary phase information to estimate the job's execution time. Our work exploits selectivity estimation and does not require such pre-execution or debug runs of MapReduce queries.

In [22], Mullin proposed a partial bloom filter based join selectivity estimation algorithm. In [6], Bell et. al presented a piece-wise uniform approach for estimating the frequency distribution of join attributes by equal-width histograms. Our paper can utilize their method for join selectivity estimation without the uniform and independent distribution assumption. In [8], Dell'aquila exploited the canonical coefficient parameters of estimating selectivity factors for relational operations through approximating both the multivariate data distribution and distinct values of attributes. Swami et al. [26] attempted selectivity prediction for multi-join operations with a common key and uniform key distribution. Our work extends this prior study to provide selectivity prediction for multi-join operations on different keys with piece-wise uniform distribution in the MapReduce environment.

Wu et al. [32] proposed AQUA as a comprehensive cost model to estimate CPU, network and I/O costs of database operations and MapReduce jobs. Our work is different from AQUA as a time based model and includes selectivity estimation for different types of query jobs. Li et al. [19] estimated query's resource demands using statistical models for individual operators of database queries. Ganapathi et al. [10, 11] developed a KCCA (Kernel Canonical Correlation Analysis) model based prediction system to solve the database query time estimation. Compared to these studies, we design a multivariate model for queries in MapReduce-based data warehouse systems.

Algorithms for scheduling jobs and/or DAGs of jobs have been studied based on general models. For example, the Johnson's algorithm [16] was proposed to solve two- and three-stage Job-shop problems. HLFET was proposed by Adam et al. [4] as a scheduling algorithm for DAGs of jobs in a multiprocessor environment. Similarly, Hu et al. [14] proposed a polynomial schedule algorithm for in-tree structured DAGs with unit computation cost. These algorithms cannot be directly applied to schedule analytic Hive queries due to the special features of Hive queries and frameworks. For example, Johnson's algorithm is not applicable for query scheduling due to the precedence constraint of jobs. In addition, parallelism level in MapReduce environment is flexible and job nodes in the query DAG are of non-uniform costs, which are different from the scenarios in [4] and [14].

Targeting the scheduling of individual MapReduce jobs, various algorithms were proposed. For example, Zaharia et al. [34] proposed delay scheduling to promote data locality in the scheduling of MapReduce jobs. Wolf et al. proposed a malleable scheduler *Flex* for optimizing the minisum and minimax metrics of response time, stretch, deadlines, etc [31]. They are not aware of the relationship of jobs in a DAG. Luo et al. [20] identified and transmitted critical semantic information from DBMS down to the hybrid storage layer, enabling the adoption of different QoS policies for different queries. In some runtime engines (e.g. Spark [35], Dryad [15], Tez [2]), the structures of DAGs are considered. Shenker et al. [25] implemented Shark as a scalable SQL and Rich Analytics on top of Spark. Yu et al. [33] designed the DryadLINQ model for users to conduct declarative operations on distributed datasets. A DryadLINQ program is translated into an execution plan graph where each vertex is to execute as a Dryad job on the cluster-computing infrastructure. Ke et al. [17] provided a framework called Optimus for dynamically rewriting EPG (Execution Plan Graph) at runtime in DryadLINQ. Compared to these studies, our work leverages semantic information from DAG queries for modeling of job execution time and resource usage and then employs the model for efficiency query scheduler.

7 CONCLUSION

Many popular data warehouse systems are deployed on top of MapReduce. Complex analytic queries are usually compiled into directed acyclic graphs (DAGs). However, the simplistic job-level scheduling policy in MapReduce is unable to balance resource distribution and reconcile the dynamic needs of different jobs in DAGs. To address such issues systematically, we develop a semantic-aware query prediction framework which includes three main techniques: semantic percolation, selectivity estimation and multivariate time prediction for analytic queries. Our framework is able to bridge the semantic gap between Hadoop and Hive. In addition, the multivariate query prediction can accurately predict the changing data sizes and the execution time of jobs in DAG queries. The query prediction framework offers important inputs for scheduling decisions in Hadoop scheduler. Experiment results demonstrate that our semantic-aware framework can achieve accurate query prediction and enable more efficient query scheduling than traditional Hadoop schedulers.

Acknowledgments

This work is funded in part by the U.S. National Science Foundation awards 1561041 and 1564647.

REFERENCES

[1] [n. d.]. Apache Hadoop Project. http://hadoop.apache.org/.
[2] [n. d.]. Apache Tez. http://hortonworks.com/hadoop/tez/.
[3] [n. d.]. TPC. http://www.tpc.org/.
[4] Thomas L Adam, K. Mani Chandy, and JR Dickson. 1974. A comparison of list schedules for parallel processing systems. *Commun. ACM* 17, 12 (1974), 685–690.
[5] Ganesh Ananthanarayanan, Ali Ghodsi, Andrew Wang, Dhruba Borthakur, Srikanth Kandula, Scott Shenker, and Ion Stoica. 2012. PACMan: Coordinated memory caching for parallel jobs. In *USENIX NSDI*.
[6] David A Bell, DHO Link, and S McClean. 1989. Pragmatic estimation of join sizes and attribute correlations. In *Data Engineering, 1989. Proceedings. Fifth International Conference on*. IEEE, 76–84.
[7] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (OSDI'04)*. USENIX Association, Berkeley, CA, USA, 10–10.
[8] Carlo DellâĂŹaquila, Ezio Lefons, and Filippo Tangorra. 2005. Analytic-based estimation of query result sizes. In *Proceedings of the 4th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering Data Bases*. WSEAS, 24.
[9] Jennie Duggan, Ugur Cetintemel, Olga Papaemmanouil, and Eli Upfal. 2011. Performance prediction for concurrent database workloads. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 337–348.
[10] Archana Ganapathi, Yanpei Chen, Armando Fox, Randy Katz, and David Patterson. 2010. Statistics-driven workload modeling for the cloud. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*. IEEE, 87–92.
[11] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L Wiener, Armando Fox, Michael Jordan, and David Patterson. 2009. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, 592–603.
[12] John Gantz and David Reinsel. 2012. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the Future* (2012).
[13] Alan F Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. 2009. Building a high-level dataflow system on top of Map-Reduce: the Pig experience. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1414–1425.
[14] Te C Hu. 1961. Parallel sequencing and assembly line problems. *Operations research* 9, 6 (1961), 841–848.
[15] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. [n. d.]. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys '07)*. 59–72.
[16] Selmer Martin Johnson. 1954. Optimal two-and three-stage production schedules with setup times included. In *Naval research logistics quarterly*, Vol. 1. Wiley Online Library, 61–68.
[17] Qifa Ke, Michael Isard, and Yuan Yu. 2013. Optimus: a dynamic rewriting framework for data-parallel execution plans. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 15–28.
[18] Rubao Lee, Tian Luo, Yin Huai, Fusheng Wang, Yongqiang He, and Xiaodong Zhang. 2011. Ysmart: Yet another sql-to-mapreduce translator. In *Distributed Computing Systems (ICDCS), 31st International Conference on*. IEEE, 25–36.
[19] Jiexing Li, Arnd Christian König, Vivek Narasayya, and Surajit Chaudhuri. 2012. Robust estimation of resource consumption for sql queries using statistical techniques. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1555–1566.
[20] Tian Luo, Rubao Lee, Michael Mesnier, Feng Chen, and Xiaodong Zhang. 2012. hStorage-DB: heterogeneity-aware data management to exploit the full capability of hybrid storage systems. *Proceedings of the VLDB Endowment* 5, 10 (2012), 1076–1087.
[21] Kristi Morton, Magdalena Balazinska, and Dan Grossman. 2010. ParaTimer: a progress indicator for MapReduce DAGs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 507–518.
[22] James K Mullin. 1993. Estimating the size of a relational join. *Information Systems* 18, 3 (1993), 189–196.
[23] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. Pig latin: a not-so-foreign language for data processing. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, New York, NY, USA, 1099–1110. https://doi.org/10.1145/1376616.1376726
[24] Gregory Piatetsky-Shapiro and Charles Connell. 1984. Accurate estimation of the number of tuples satisfying a condition. In *ACM SIGMOD Record*, Vol. 14. ACM, 256–276.
[25] Scott Shenker, Ion Stoica, Matei Zaharia, Reynold Xin, Josh Rosen, and Michael J Franklin. 2013. Shark: SQL and Rich Analytics at Scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*.
[26] A. Swami and K.B. Schiefer. 1993. On the estimation of join result sizes. *IBM Technical Report* (1993).
[27] Jian Tan, Xiaoqiao Meng, and Li Zhang. 2012. Delay tails in MapReduce scheduling. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems (SIGMETRICS '12)*. ACM, New York, NY, USA, 5–16. https://doi.org/10.1145/2254756.2254761
[28] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang 0002, Suresh Anthony, Hao Liu, and Raghotham Murthy. 2010. Hive - a petabyte scale data warehouse using Hadoop. In *ICDE*. 996–1005.
[29] Abhishek Verma, Ludmila Cherkasova, and Roy H Campbell. 2011. ARIA: automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 235–244.
[30] Yandong Wang, Jian Tan, Weikuan Yu, Xiaoqiao Meng, and Li Zhang. 2013. Preemptive reducetask scheduling for fair and fast job completion. In *Proceedings of the 10th International Conference on Autonomic Computing, ICAC*, Vol. 13.
[31] Joel Wolf, Deepak Rajan, Kirsten Hildrum, Rohit Khandekar, Vibhore Kumar, Sujay Parekh, Kun-Lung Wu, and Andrey Balmin. 2010. Flex: A slot allocation scheduling optimizer for mapreduce workloads. In *Middleware'10*. Springer, 1–20.
[32] Sai Wu, Feng Li, Sharad Mehrotra, and Beng Chin Ooi. 2011. Query optimization for massively parallel data processing. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 12.
[33] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. 2008. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language.. In *OSDI*, Vol. 8. 1–14.
[34] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems (EuroSys'10)*. ACM, New York, NY, USA, 265–278. https://doi.org/10.1145/1755913.1755940
[35] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. 10–10.
[36] Zhuoyao Zhang, Ludmila Cherkasova, Abhishek Verma, and Boon Thau Loo. 2012. Automated profiling and resource management of pig programs for meeting service level objectives. In *Proceedings of the 9th international conference on Autonomic computing*. ACM, 53–62.