Optimizing Completion Time and Resource Provisioning of Pig Programs

Zhuoyao Zhang
University of Pennsylvania
Philadelphia, PA, USA
zhuoyao@seas.upenn.edu

Ludmila Cherkasova
Hewlett-Packard Labs
Palo Alto, CA, USA
lucy.cherkasova@hp.com

Abhishek Verma
University of Illinois
at Urbana-Champaign, IL, USA
verma7@illinois.edu

Boon Thau Loo
University of Pennsylvania
Philadelphia, PA, USA
boonloo@cis.upenn.edu

***Abstract*—As cloud computing continues to mature, IT managers have started concentrating on the support of additional performance requirements: quality of service and tailored resource allocation for achieving service performance goals. In this paper[1], we consider the popular Pig framework that provides a high-level SQL-like abstraction on top of MapReduce engine for processing large data sets. Programs written in such frameworks are compiled into directed acyclic graphs (DAGs) of MapReduce jobs. Often, data processing applications have to produce results by a certain time deadline. We design a performance modeling framework for Pig programs that solves two inter-related problems: (i) estimating the completion time of a Pig program as a function of allocated resources; (ii) estimating the amount of resources (a number of map and reduce slots) required for completing a Pig program with a given (soft) deadline. To achieve these goals, we first, optimize a Pig program execution by enforcing the *optimal schedule* of its concurrent jobs. This optimization reduces a program completion time (10%-27% in our experiments), and moreover, it eliminates possible non-determinism in the DAGs execution. Based on our optimization, we propose an accurate performance model for Pig programs. This approach leads to significant resource savings (20%-60% in our experiments) compared with the original, unoptimized solution. We validate our approach in a 66-node Hadoop cluster using two workload sets: TPC-H queries and a set of customized queries mining a collection of HP Labs' web proxy logs.**

***Keywords*-MapReduce, Pig, performance model, resource allocation, execution optimization, job scheduling.**

I. Introduction

With the increasing popularity of cloud computing, many companies are now moving towards the use of cloud infrastructures to quickly process large quantities of new data to drive their core business. MapReduce [1] and its open-source implementation Hadoop offer a scalable and fault-tolerant framework for processing large data sets. To enable programmers to specify more complex queries in an easier way, several projects, such as Pig [2], Hive [3], Scope [4], and Dryad [5], provide high-level SQL-like abstractions on top of MapReduce engines. In these frameworks, complex analytics tasks are expressed as high-level declarative abstractions and then they are compiled into *directed acyclic graphs* (DAGs) of MapReduce jobs. There is a growing number of MapReduce applications, e.g., personalized advertising, sentiment analysis, spam and fraud detection, real-time event log analysis, etc., that require completion time guarantees, i.e., they have specific Service Level Objectives (SLOs) and are deadline-driven.

[1]This work was largely completed during Z. Zhang's and A. Verma's internship at HP Labs. B. T. Loo and Z. Zhang are supported in part by NSF grants (CNS-1117185, CNS-0845552 , IIS-0812270). A. Verma is supported in part by NSF grant CCF-0964471.

One of the key challenges in the cloud environment is the need to manage an SLO-driven *resource allocation* of cloud resources shared across multiple users. While there have been some research efforts [6], [7] towards developing performance models for MapReduce jobs, these techniques do not apply to complex queries consisting of MapReduce DAGs. To address this limitation, we consider the popular Pig framework [2] for *solving the following problems*: (i) estimate the completion time of a Pig program as a function of allocated resources; (ii) estimate the amount of resources (a number of map and reduce slots) required for completing a Pig program with a given (soft) deadline.

For a Pig program defined by a DAG of MapReduce jobs, its completion time might be approximated as the sum of completion times of the jobs that constitute this Pig program. However, such model might lead to a higher time estimate than the actual measured program time. The reason is that unlike the execution of sequential jobs where the next job can only start after the previous one is completed, for concurrent jobs, their executions may "overlap" in time. The performance model should take this "overlap" in executions of concurrent jobs into account. Moreover, the execution order of concurrent jobs in the Pig program may impact the program processing time. Using this observation, we first, optimize a Pig program execution by enforcing the *optimal schedule* of its concurrent jobs. We evaluate optimized Pig programs and the related performance improvements using TPC-H queries and a set of customized queries mining a collection of HP Labs' web proxy logs (both sets are presented by the DAGs with concurrent jobs). Our results show 10%-27% decrease in Pig program completion times.

The proposed Pig optimization has another useful outcome: it eliminates existing non-determinism in Pig program execution of concurrent jobs, and therefore, it enables better performance predictions. We develop an accurate performance model for completion time estimates and resource allocations of optimized Pig programs. The accuracy of this model is validated using a combination of TPC-H and web proxy log analysis queries. We show that for Pig programs with concurrent jobs, this approach leads to significant resource savings (20%-60% in our experiments) compared with the original, non-optimized solution.

While our work is based on the Pig experience, we believe that the proposed model and optimizations are general and can be applied for performance modeling and resource allocations of complex analytics tasks that are expressed as an ensemble (DAG) of MapReduce jobs.

This paper is organized as follows. Section II provides a background on the Pig framework. Section III discusses

subtleties of concurrent jobs execution in Pig, introduces optimized scheduling of concurrent jobs, and offers a Pig performance model. The accuracy of the model is evaluated in Section IV. Section V describes the related work. Section VI presents a summary and future directions.

II. Background: Pig Programs

There are two main components in the Pig system:

- The *language*, called Pig Latin, that combines high-level declarative style of SQL and the low-level procedural programming of MapReduce. A Pig program is similar to specifying a query execution plan: it represents a sequence of steps, where each one carries a single data transformation using a high-level data manipulation constructs, like *filter*, *group*, *join*, etc.
- The *execution environment* to run Pig programs. The Pig system takes a Pig Latin program as input, compiles it into a DAG of MapReduce jobs, and coordinates their execution on a given Hadoop cluster.

The following specification shows a simple example of a Pig program. It describes a task that operates over a table *URLs* that stores data with the three attributes: *(url, category, pagerank)*. This program identifies for each *category* the *url* with the highest *pagerank* in that *category*.

URLs = ***load*** *'dataset' as (url, category, pagerank);*
groups = ***group*** *URLs by category;*
result = ***foreach*** *groups generate group, max(URLs.pagerank);*
store *result into 'myOutput'*

The example Pig program is compiled into a single MapReduce job. Typically, Pig programs are more complex, and can be compiled into an execution plan consisting of several stages of MapReduce jobs, some of which can run concurrently. Figure 1 shows a possible DAG of five MapReduce jobs $\{j_1, j_2, j_3, j_4, j_5\}$, where each node represents a MapReduce job, and the edges between the nodes represent the *data dependencies* between jobs.

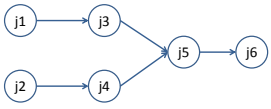


Figure 1. Example of a Pig program' execution plan represented as a DAG of MapReduce jobs.

To execute the plan, the Pig engine will first submit all the *ready* jobs (i.e., the jobs that do not have data dependency on the other jobs) to Hadoop. After Hadoop has processed these jobs, the Pig system will delete those jobs and the corresponding edges from the processing DAG, and will submit the next set of ready jobs. This process continues until all the jobs are completed. In this way, the Pig engine partitions the DAG into multiple stages, each containing one or more independent MapReduce jobs that can be executed concurrently. For example, the DAG shown in Figure 1 will be partitioned into the following four stages for processing:

- first stage: $\{j_1, j_2\}$;
- second stage: $\{j_3, j_4\}$;
- third stage: $\{j_5\}$;
- fourth stage: $\{j_6\}$.

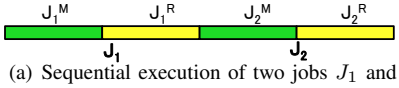
Note that for stages with concurrent jobs, there is no specifically defined ordering in which the jobs are going to be executed by Hadoop.

III. Performance Model for Pig Programs

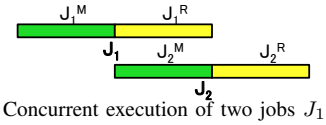
In this section, we analyze subtleties in execution of concurrent MapReduce jobs and demonstrate that the job order has a significant impact on the program completion time. We optimize a Pig program by enforcing the *optimal schedule* of its concurrent jobs. Then we introduce an accurate performance modeling framework for Pig programs.

A. Modeling Concurrent Jobs' Executions

Let us consider two concurrent MapReduce jobs J_1 and J_2 and how they are going to be executed by the Hadoop cluster with a default FIFO scheduler. Let us also assume that there are no data dependencies among the concurrent jobs. Therefore, unlike the execution of sequential jobs where the next job can only start after the previous one is entirely finished (shown in Figure 2 (a)), for concurrent jobs, once the previous job completes its map phase (map phase is represented by the green color), releases map slots, and begins reduce phase processing (reduce phase is represented by the yellow color) , the next job can start its map phase execution with the released map resources in a pipelined fashion (shown in Figure 2 (b)). The Pig performance model should take this "overlap" in executions of concurrent jobs into account.



(a) Sequential execution of two jobs J_1 and J_2.



(b) Concurrent execution of two jobs J_1 and J_2.

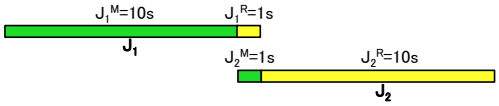
Figure 2. Difference in executions of (a) two sequential MapReduce jobs; (b) two concurrent MapReduce jobs.

We find one more interesting observation about concurrent jobs' execution of the Pig program. The original Hadoop implementation executes concurrent MapReduce jobs from the same Pig program in a random order. Some ordering may lead to a significantly less efficient resource usage and an increased processing time. Consider the following example with two concurrent MapReduce jobs:

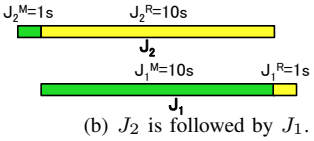
- Job J_1 has a map phase duration of $J_1^M = 10s$ and the reduce phase duration of $J_1^R = 1s$.
- Job J_2 has a map phase duration of $J_2^M = 1s$ and the reduce phase duration of $J_2^R = 10s$.

There are two possible executions shown in Figure 3:

- J_1 is followed by J_2 shown in Figure 3(a). The reduce phase of J_1 overlaps with the map phase of J_2 leading to overlap of only $1s$. The total completion time of processing two jobs is $10s + 1s + 10s = 21s$.
- J_2 is followed by J_1 shown in Figure 3(b). The reduce phase of J_2 overlaps with the map phase of J_1 leading to a much better pipelined execution and a larger overlap of 10s. The total makespan is $1s + 10s + 1s = 12s$.



(a) J_1 is followed by J_2.



(b) J_2 is followed by J_1.

Figure 3. Impact of concurrent job scheduling on their completion time.

There is a significant difference in the job completion time (75% in the example above) depending on the execution order of the jobs. We optimize a Pig program execution by enforcing the optimal schedule of its concurrent jobs. We apply the classic Johnson algorithm for building the optimal two-stage jobs' schedule [8]. The optimal execution of concurrent jobs leads to improved completion time. Moreover, this optimization eliminates possible non-determinism in Pig program execution, and enables more accurate completion time predictions for Pig programs.

B. Completion Time Estimates for Pig Programs

As a building block for modeling a Pig program defined as a DAG of MapReduce jobs, we apply the approach introduced in ARIA [6] for performance modeling of a single MapReduce job. We extract performance profiles of all the jobs in the DAG from the past program executions. Using these job profiles we can predict the completion time of each job (and completion time of map and reduce phases) as a function of allocated map and reduce slots.

Let us consider a Pig program P that is compiled into a DAG of MapReduce jobs and consists of S *stages*. The Pig engine partitions the DAG into multiple stages that are executed one after another, where each stage contains one or more independent MapReduce jobs which can be executed concurrently. Note that due to data dependencies within a Pig execution plan, the *next stage* cannot start until the *previous stage* finishes. Let T_{S_i} denote the completion time of stage S_i. Thus, the completion of a Pig program P can be estimated as follows:

$$T_P = \sum_{1 \le i \le S} T_{S_i}. \quad (1)$$

For a stage that consists of a single job J, the stage completion time is defined by the job J's completion time.

For a stage that contains concurrent jobs, the stage completion time depends on the jobs' execution order. Suppose there are $|S_i|$ jobs within a particular stage S_i and the jobs are executed according to the order $\{J_1, J_2, ...J_{|S_i|}\}$. Note, that given a number of allocated map/reduce slots (S_M^P, S_R^P) to the Pig program P, we can compute for any MapReduce job $J_i (1 \le i \le |S_i|)$ the duration of its map and reduce phases that are required for the Johnson's algorithm [8] to determine the optimal schedule of the jobs $\{J_1, J_2, ...J_{|S_i|}\}$.

Let us assume, that for each stage with concurrent jobs, we have already determined the optimal job schedule that minimizes the completion time of the stage. Now, we introduce the performance model for predicting the Pig program P completion time T_P as a function of allocated resources (S_M^P, S_R^P). We use the following notations:

$timeStart_{J_i}^M$	the start time of job J_i's map phase
$timeEnd_{J_i}^M$	the end time of job J_i's map phase
$timeStart_{J_i}^R$	the start time of job J_i's reduce phase
$timeEnd_{J_i}^R$	the end time of job J_i's reduce phase

Then the stage completion time can be estimated as

$$T_{S_i} = timeEnd_{J_{|S_i|}}^R - timeStart_{J_1}^M \quad (2)$$

We now explain how to estimate the start/end time of each job's map/reduce phase. Let $T_{J_i}^M$ and $T_{J_i}^R$ denote the completion times of map and reduce phases of job J_i respectively. Then

$$timeEnd_{J_i}^M = timeStart_{J_i}^M + T_{J_i}^M \quad (3)$$
$$timeEnd_{J_i}^R = timeStart_{J_i}^R + T_{J_i}^R \quad (4)$$

Figure 4 shows an example of three concurrent jobs execution in the order J_1, J_2, J_3.

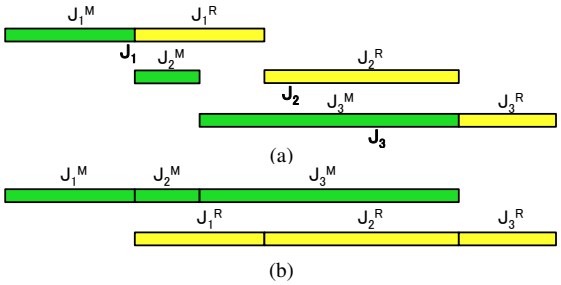


Figure 4. Execution of Concurrent Jobs

Note, that Figure 4 (a) can be rearranged to show the execution of jobs' map/reduce stages separately (over the map/reduce slots) as shown in Figure 4 (b). It is easy to see that since all the concurrent jobs are independent, the map phase of the next job can start immediately once the previous job's map stage is finished, i.e.,

$$timeStart_{J_i}^M = timeEnd_{J_{i-1}}^M = timeStart_{J_{i-1}}^M + T_{J_{i-1}}^M \quad (5)$$

The start time $timeStart_{J_i}^R$ of the reduce stage of the concurrent job J_i should satisfy the following two conditions:

1) $timeStart_{J_i}^R \ge timeEnd_{J_i}^M$
2) $timeStart_{J_i}^R \ge timeEnd_{J_{i-1}}^R$

Therefore, we have the following equation:

$$timeStart_{J_i}^R = max\{timeEnd_{J_i}^M, timeEnd_{J_{i-1}}^R\} =$$
$$= max\{timeStart_{J_i}^M + T_{J_i}^M, timeStart_{J_{i-1}}^R + T_{J_{i-1}}^R\} \quad (6)$$

Finally, the completion time of the entire Pig program P is defined as the *sum of its stages* using eq. (1).

C. Resource Allocation Estimates for Pig Programs

Let us consider a Pig program P with a given deadline D. The optimized execution of P may significantly improve the program completion time. Therefore, P may need to be

assigned a smaller amount of resources for meeting a given deadline D compared to its non-optimized execution.

First, we explain how to approximate the resource allocation of a non-optimized execution of a Pig program. The completion time of non-optimized P can be represented as a sum of completion time of the jobs that comprise the DAG of this Pig program. Thus, for a Pig program P that contains $|P|$ jobs, its completion time can be estimated as a function of assigned map and reduce slots (S_M^P, S_R^P) as follows:

$$T_P(S_M^P, S_R^P) = \sum_{1 \le i \le |P|} T_{J_i}(S_M^P, S_R^P) \quad (7)$$

The unique benefit of this model is that it allows us to express the completion time D of a Pig program P via a special form equation shown below:

$$D = \frac{A^P}{S_M^P} + \frac{A^P}{S_R^P} + C^P \quad (8)$$

This equation can be used for solving the inverse problem of finding resource allocations (S_M^P, S_R^P) such that P completes within time D. This equation yields a hyperbola if (S_M^P, S_R^P) are considered as variables. We can directly calculate the minima on this curve using Lagrange's multipliers similarly as proposed in ARIA [6] for finding the resource allocation of a single MapReduce job with a given deadline.

The performance model introduced in the previous Section III-B for accurate completion time estimates of an optimized Pig program is more complex. It requires computing a function *max* for stages with concurrent jobs, and therefore, it cannot be expressed as a single equation for solving the inverse problem of finding the appropriate resource allocation. However, we can use the "over-provisioned" resource allocation defined by eq. (8) as an initial point for determining the solution required by the optimized Pig program P. The hyperbola with all the possible solutions according to the "over-sized" model is shown in Figure 5 as the red curve, and $A(M, R)$ represents the point with a minimal number of map and reduce slots (i.e., the pair (M, R) results in the minimal sum of map and reduce slots). We designed the following algorithm described below that determines the minimal resource allocation pair (M_{min}, R_{min}) for an optimized Pig program P with deadline D. This computation is illustrated by Figure 5.

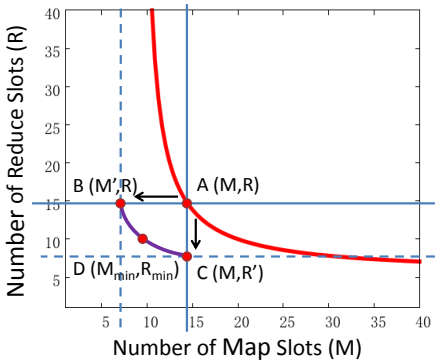


Figure 5. Resource allocation estimates for an optimized Pig program.

First, we find the minimal number of map slots M' (i.e., the pair (M', R)) such that deadline D can still be met by the optimized Pig program with the enforced optimal execution of its concurrent jobs. We do it by fixing the number of reduce slots to R, and then step-by-step reducing the allocation of map slots. Specifically, our algorithm sets the resource allocation to $(M-1, R)$ and checks whether program P can still be completed within time D. If the answer is positive, then it tries $(M-2, R)$ as the next allocation. This process continues until point $B(M', R)$ (see Figure 5) is found such that the number M' of map slots cannot be further reduced for meeting a given deadline D.

At the second step, we apply the same process for finding the minimal number of reduce slots R' (i.e., the pair (M, R')) such that the deadline D can still be met by the optimized Pig program P.

At the third step, we determine the intermediate values on the curve between (M', R) and (M, R') such that deadline D is met by the optimized Pig program P. Starting from point (M', R), we are trying to find the allocation of map slots from M' to M, such that the minimal number of reduce slots $\hat{R}$ should be assigned to P for meeting its deadline.

Finally, (M_{min}, R_{min}) is the pair on this curve such that it results in the the minimal sum of map and reduce slots.

IV. Evaluation of the Pig Performance Model

In this section, we evaluate performance benefits of introduced Pig program optimization and assess the accuracy of the proposed performance model.

A. Experimental Testbed and Workload

All experiments are performed on 66 HP DL145 GL3 machines. Each machine has four AMD 2.39GHz cores, 8 GB RAM and two 160GB hard disks. The machines are set up in two racks and interconnected with gigabit Ethernet. We used Hadoop 0.20.2 and Pig-0.7.0 with two machines dedicated as the JobTracker and the NameNode, and remaining 64 machines as workers. Each worker is configured with 2 map and 1 reduce slots. The file system blocksize is set to 64MB. The replication level is set to 3. We disabled speculative execution since it did not lead to significant improvements in our experiments.

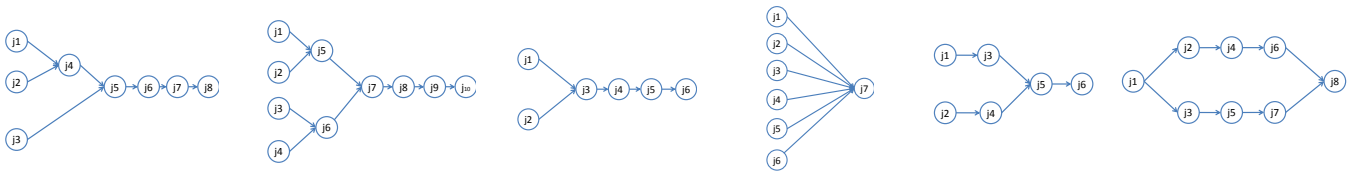
In our case study, we use the following two workloads [2]:

TPC-H. This workload is based on TPC-H [10], a standard database benchmark for decision-support workloads. We select three queries $Q5, Q8, Q10$ out of existing 22 SQL queries and express them as Pig programs. The input dataset size is 9GB (scaling factor 9 using the standard data generator). For each query, we select a logical plan that results in a DAG of concurrent MapReduce jobs shown in Figures 6 (a),(b),(c) respectively[3]:

- The *TPC-H Q5* query joins 6 tables, and its dataflow results in 3 concurrent MapReduce jobs.

[2]We did not use a popular PigMix benchmark [9] because only 1 of 17 Pig programs of the benchmark contains concurrent jobs.

[3]While more efficient logical plans may exist, our goal here is to create a DAG with concurrent jobs to stress test our model.



(a) *TPC-H Q5* (b) *TPC-H Q8* (c) *TPC-H Q10* (d) *Proxy Q1* (e) *Proxy Q2* (f) *Proxy Q3*

Figure 6. DAGs of Pig programs in the TPC-H and HP Labs Proxy query sets.

- The *TPC-H Q8* query joins 8 tables, and its dataflow results in two stages with 4 and 2 concurrent jobs.
- The *TPC-H Q10* query joins 4 tables, and its dataflow results in 2 concurrent MapReduce jobs.

HP Labs' Web Proxy Query Set. This workload consists of a set of Pig programs for analyzing HP Labs' web proxy logs. It contains 6 months access logs to web proxy gateway at HP Labs. The total dataset size (6 months) is about 18 GB (438 million records). The fields include information such as *date, time, time-taken, c-ip, cs-host*, etc. We intend to create realistic Pig queries executed on real-world data.

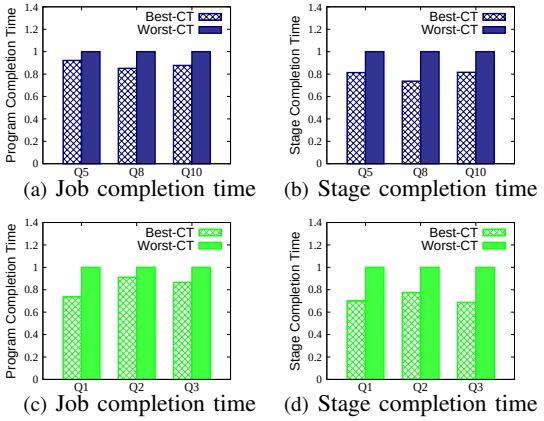
- The *Proxy Q1* program investigates the dynamics in access frequencies to different websites per month and compares them across the 6 months. The Pig program results in 6 concurrent MapReduce jobs with the DAG of the program shown in Figure 6 (d).
- The *Proxy Q2* program tries to discover the co-relationship between two websites from different sets (tables) of popular websites: the first set is created to represent the top 500 popular websites accessed by web users within the enterprise. The second set contains the top 100 popular websites in US according to Alexa's statistics (http://www.alexa.com/topsites). The program DAG is shown in Figure 6 (e).
- The *Proxy Q3* program presents the intersect of 100 most popular websites (i.e., websites with highest access frequencies) accessed both during work and after work hours. The DAG of the program is shown in Figure 6 (f).

B. Optimal Schedule of Concurrent Jobs

Figure 7 shows the impact of concurrent jobs scheduling on the completion time of TPC-H and Proxy queries when each program is processed with **128** map and **64** reduce slots.

Figures 7 (a) and (c) show two extreme measurements: the best program completion time (i.e., when the optimal schedule of concurrent jobs is chosen) and the worst one (i.e., when concurrent jobs are executed in the "worst" possible order based on our estimates). For presentation purposes, the best (optimal) completion time time is **normalized** with respect to the worst one. The choice of optimal schedule of concurrent jobs reduces the completion time by 10%-27% compared with the worse case ordering.

Figures 7 (b) and (d) show completion times of stages with concurrent jobs under different schedules for the same TPC-H and Proxy queries. Performance benefits at the stage level are even higher: they range between 20%-30%.



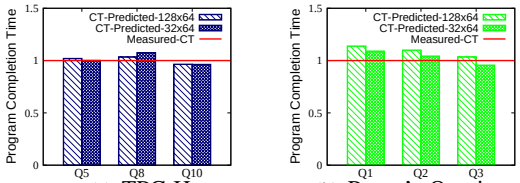
(a) Job completion time (b) Stage completion time

(c) Job completion time (d) Stage completion time

Figure 7. Normalized completion time for different schedules of concurrent jobs: (a-b) TPC-H, (c-d) HP Labs proxy queries.

C. Predicting Completion Time and Required Resource Allocation of Optimized Pig Programs

Figure 8 shows the Pig program completion time estimates based on the proposed performance model for TPC-H and Proxy queries. Figure 8 shows the results when each program is processed with **128x64** and **32x64** map and reduce slots respectively.



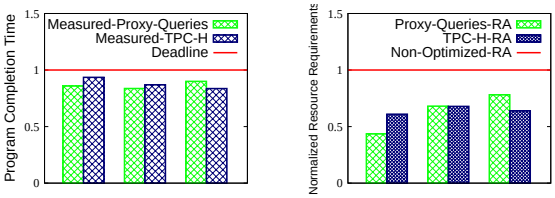
(a) TPC-H (b) Proxy's Queries

Figure 8. Predicted Pig programs completion times.

In most cases (11 out of 12), the predicted completion times are within 10% of the measured ones.

Let T denote the Pig program completion time when program P is processed with maximum available cluster resources. We set $D = 2 \cdot T$ as a completion time goal. Then we compute the required resource allocation for P to meet the deadline D. Figure 9 (a) shows measured completion times achieved by the TPC-H and Proxy's queries respectively when they are assigned the resource allocations computed with the designed resource allocation model. All the queries complete within 10% of the targeted deadlines.

Figure 9 (b) compares the amount of resources (the sum of map and reduce slots) for non-optimized and optimized executions of TPC-H and Proxy's queries respectively. The



(a) Can we meet deadlines? (b) Resource requirements of optimized Pig programs.

Figure 9. Resource allocations for optimized Pig programs.

optimized executions are able to achieve targeted deadlines with much smaller resource allocations (20%-60% smaller) compared to resource allocations for non-optimized Pig programs. Therefore, the proposed optimal schedule of concurrent jobs leads to significant resource savings for deadline-driven Pig programs.

V. Related work

While performance modeling in the MapReduce framework is a new topic, there are several interesting research efforts in this direction. Researchers designed job scheduling for MapReduce/Hadoop framework such as FLEX [7], ARIA [6], etc. Typically, these schedulers utilize a performance model of a *single MapReduce job* execution.

Starfish [11] applies *dynamic instrumentation* to collect a detailed run-time monitoring information about job execution at a fine granularity: data reading, map processing, spilling, merging, shuffling, sorting, reduce processing and writing. The authors offer a workflow-aware scheduler that correlate data (block) placement with task scheduling to optimize the workflow completion time. In our work, we propose complementary optimizations based on optimal scheduling of concurrent jobs within the DAG to minimize overall completion time.

Tian and Chen [12] aim to predict performance of a single MapReduce program from the test runs with a smaller number of nodes. They consider MapReduce processing at a fine granularity. The authors use a *linear regression technique* to approximate the cost (duration) of each processing function. The problem of finding resource allocations that support given job completion goals are formulated as an optimization problem that can be solved with existing commercial solvers.

CoScan [13] offers a special scheduling framework that merges the execution of Pig programs with common data inputs in such a way that this data is only scanned once. Authors augment Pig programs with a set of *(deadline, reward)* options to achieve. Then they formulate the schedule as an optimization problem and offer a heuristic solution.

Morton et al. [14] propose *ParaTimer*: the progress estimator for parallel queries expressed as Pig scripts [2]. The approach is based on precomputing the expected schedule of all the tasks, and therefore identifying all the pipelines (sequences of MapReduce jobs) in the query. However, this work does not provide a technique for estimating the amount of resources (a number of map and reduce slots) required for completing a Pig program with a given (soft) deadline.

VI. Conclusion

Efficient Hadoop management requires new performance tools to navigate SLO-driven job scheduling and tailored resource allocation in the shared cluster. In our work, we have introduced a novel performance modeling framework for processing Pig programs with deadlines that does not require any modifications or instrumentation of either the application or underlying Hadoop/Pig execution engines. The proposed approach offers an optimized scheduling of concurrent jobs within a DAG that allows to significantly reduce the overall completion time. Our performance models are designed for the case without node failures. We see a natural extension for incorporating different failure scenarios and estimating their impact on the application performance and achievable "degraded" SLOs.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, 2008.

[2] A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava., "Building a High-Level Dataflow System on Top of Map-Reduce: The Pig Experience." *Proc. of the VLDB Endowment*, vol. 2, no. 2, 2009.

[3] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy., "Hive - a Warehousing Solution over a Map-Reduce Framework," *Proc. of VLDB*, 2009.

[4] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou., "Easy and Efficient Parallel Processing of Massive Data Sets," *Proc. of the VLDB Endowment*, vol. 1, no. 2, 2008.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," *ACM SIGOPS OS Review*, vol. 41, no. 3, 2007.

[6] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for MapReduce Environments," *Proc. of the 8th ACM International Conference on Autonomic Computing (ICAC'2011)*, 2011.

[7] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin, "FLEX: A Slot Allocation Scheduling Optimizer for MapReduce Workloads," *Proc. of ACM/IFIP/USENIX Middleware Conference*, 2010.

[8] S. Johnson., "Optimal Two- and Three-Stage Production Schedules with Setup Times Included." *Naval Res. Log. Quart.*, 1954.

[9] Apache, "PigMix Benchmark." [Online]. Available: http://wiki.apache.org/pig/PigMix

[10] "TPC Benchmark H (Decision Support), Version 2.8.0." [Online]. Available: http://www.tpc.org/tpch/

[11] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu, "Starfish: A Self-tuning System for Big Data Analytics." in *Proc. of 5th Conf. on Innovative Data Systems Research (CIDR)*, 2011.

[12] F. Tian and K. Chen, "Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds." in *Proc. of IEEE Conference on Cloud Computing, 2011*.

[13] X. Wang, C. Olston, A. Sarma, and R. Burns, "CoScan: Cooperative Scan Sharing in the Cloud," in *Proc. of the ACM Symposium on Cloud Computing,(SOCC'2011)*, 2011.

[14] K. Morton, M. Balazinska, and D. Grossman, "ParaTimer: a progress indicator for MapReduce DAGs." in *Proc. of SIGMOD*. ACM, 2010.