A Colossal Attempt to Manage Multi-tenant Big Data Analytics at Scale

Zilong Tan
Duke University
ztan@cs.duke.edu

Shivnath Babu
Duke University
shivnath@cs.duke.edu

Eric Chu
Rocket Fuel, Inc.
echu@rocketfuel.com

ABSTRACT

Efficient processing of big data has given rise to multi-tenant, big data clusters where multiple applications run on and share the same resources and data. Such multi-tenancy leads to considerable heterogeneity in the types of workloads, data sets, and users and their skills. This heterogeneity, in turn, poses significant challenges for human operators of the cluster to tune the cluster to meet performance requirements that often conflict each other. In this paper, we use operational experiences from a large, multi-tenant cluster to discuss these challenges. We describe Colossal, a platform that we developed to tackle these challenges.

1. INTRODUCTION

Big Data Brings Multi-tenancy: It has become essential for companies to process large amounts of data (popularly called *big data* today). Efficient processing of big data requires moving computation to the data, thereby requiring application code to run on the same machines where the data resides. Consequently, the need to process big data leads to multi-tenancy in which multiple applications run on and share the same resources and data [6, 11, 2, 7].

Multi-tenancy Breeds Heterogeneity: The various tenants in a multi-tenant big data deployment are seldom homogeneous in their behavior and performance requirements. Heterogeneity can arise along multiple dimensions:

- Applications may create different types of workloads, from ad-hoc, interactive queries, to workflows that recur at different frequencies, with varying complexities.
- Applications may create and analyze different data sets varying in their data types, volume, and velocity.
- Applications may use different high-level platforms (e.g., Hive, Pig), computational engines (e.g., MapReduce, Spark, Tez), workflow scheduling mechanisms (e.g., Oozie), and graph processing systems (e.g., Giraph).
- Applications may be created by a user base with a diverse set of skills and expectations, such as computer scientists, statisticians, and business analysts.
- Last, but not least, applications may have conflicting performance requirements, such as low latency, high throughput, large I/O, large amounts of memory, and rapid application development cycles.

Gatekeepers of Multi-tenancy: To meet the conflicting requirements amid different kinds of heterogeneity in a multi-tenant big data system, various mechanisms are used to control how resources are allocated to the applications sharing the cluster. A common technique is to map applications to a set of *pools*. Policies are then defined to determine: (a) How are resources distributed across different pools? (b) When is an application admitted to use resources assigned to its pool? (c) How are resources distributed across different applications in the same pool?

These policies can have a huge impact on the performance of a multi-tenant cluster. However, defining policies to achieve good performance is extremely challenging because of the conflicting requirements and the complexities of the system and multi-tenancy. Unfortunately, in practice this responsibility often falls largely on the operations team that manages the cluster, with few effective tools to make data-driven decisions.

In this paper, we use operational experiences from a large multi-tenant cluster at Rocket Fuel to discuss challenges that operators of these clusters face. Rocket Fuel has a fast-growing Hadoop cluster that currently stores approximately 30 petabytes of data across more than 1000 nodes. Almost all the above types of heterogeneity are seen in this cluster. We break down the challenges in the form of questions that these operators have to answer routinely. For example:

- *What is the effective utilization of my cluster?* We will show in Section 2 that the effective utilization of a multi-tenant cluster can be much less than what cluster operators would think based on typical monitoring tools for resource usage.
- *Should preemption be turned on?* The ability for an application X to pre-empt another application Y and start using the resources allocated to Y is one of the configurable policies in multi-tenancy; however, if not used wisely, preemption could lead to significant wasted time and resources.
- *What will be the overall impact of a 10% increase in the workload of Pool A?* We will show in Section 2 that a small increase in the workload of a pool can have a major impact on the performance of applications in another pool in a multi-tenant cluster.
- *What will be the overall impact of changing the data layout of a table T?* The application that populates T may belong to a different pool from applications that read from T; which in turn could be used to build more tables used by other applications from any pool. These dependencies make the problem of tuning data layouts in a multi-tenant cluster highly challenging.

Contributions and Roadmap: The above examples come from a large space of questions that human operators have to answer in order to manage multi-tenant clusters effectively. To the best of our

7^{th} *Biennial Conference on Innovative Data Systems Research (CIDR)* January 7-10, 2007, Asilomar, California, USA.

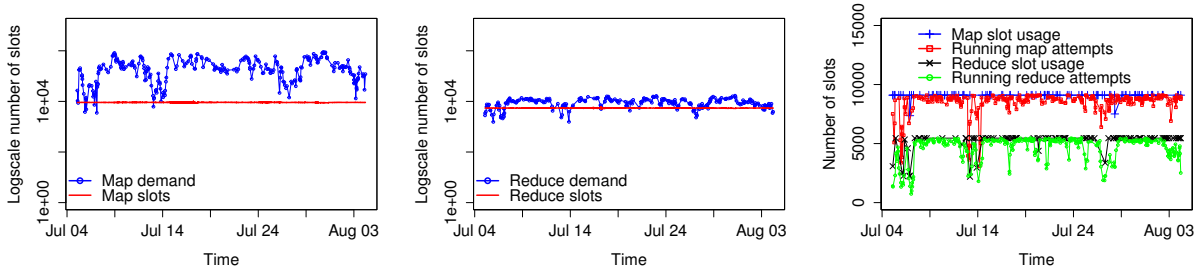

Figure 1: (a) Map demand (log-scale), (b) Reduce demand (log-scale), and (c) Resource usage on the cluster

knowledge, no single solution exists today that allows operators to ask these questions and get data-driven answers automatically. This paper presents *Colossal*, a solution that we have created for this problem.

We use real-life examples in Section 2 to motivate the challenges that operators of multi-tenant clusters face. Section 3 presents Colossal. Section 4 discusses related work and concludes the paper.

2. OPERATIONAL CHALLENGES IN MULTI-TENANT CLUSTERS FOR BIG DATA ANALYTICS

Managed by a small cluster operations team, Rocket Fuel's multi-tenant Hadoop cluster has approximately 1000 nodes storing over 30 petabytes of data. The cluster supports applications created by many teams within the company. The operations team has created six pools to manage multi-tenancy. We will call these pools ETL1, ETL2, APP1, APP2, BI, and Default.

- The ETL1 pool runs a very large number of jobs every day—many of which are low-level MapReduce jobs—that perform data copying and extract-transform-load (ETL) operations.
- The ETL2 pool contains a mixed workload that includes advanced SQL analytics, machine-learning computations, ad-hoc queries, and repeated workflows.
- The APP pools contain ad-hoc or repeated applications from different teams.
- The BI pool contains ad-hoc queries in SQL with considerable use of user-defined functions.
- The Default pool is for applications that are not mapped to one of the other pools.

Assignment of applications to these pools is done statically. Resources are partitioned dynamically among the pools based on the fair share resource allocation policy as implemented by Hadoop's *Fair* scheduler [3]. As we will discuss later, how resources are partitioned across pools and within a pool are based on configurations defined statically by the operations team.

Broadly speaking, the objective of the cluster operations team is to maximize cluster performance while meeting the business service-level agreements (SLAs). However, as part of the overall multi-tenant resource management process, the cluster operations team has to serve several, potentially conflicting, interests. For example, users such as business analysts (who use the BI pool) wish to minimize the latency of their jobs; and show little interest in the overall performance of the cluster. The manager of a team that is responsible for a repeated workflow (e.g., one that refreshes a base table used by many time-sensitive analysis tasks) will only care about ensuring that this workflow gets enough resources to finish in time. This workflow may be part of a number of applications that share the ETL2 pool.

Multi-tenant scenarios like what we see at Rocket Fuel are now common in companies that process large amounts of data. The larger heterogeneity (recall Section 1) and the more democratic nature of Hadoop usage bring unprecedented operational challenges compared to managing multi-tenancy in traditional parallel database systems. In the rest of this section, we aim to create a deeper understanding of the problems in multi-tenant resource management. We will use real examples of problems seen at Rocket Fuel. In particular, we will look at these problems from the perspective of Alice. Alice is a fictional member of the cluster operations team who is responsible for managing multi-tenancy in the cluster.

2.1 Understanding Effective Cluster Utilization

Figures 1(a) and 1(b) show graphs that Alice looks at routinely in order to understand the overall demand for resources on the cluster. In Hadoop, the resources in the cluster are discretized linearly into *slots*. The cluster contains a total *MAP_SLOTS* number of *map slots* and *REDUCE_SLOTS* number of *reduce slots*. A map (reduce) slot can run one map (reduce) attempt at any point of time.

Figure 1(a) (Figure 1(b)) plots the *demand* for map (reduce) slots in comparison to the total number of map (reduce) slots available. This data is for the time from July 5 to August 4, 2014. The map (reduce) demand at a time t is defined as the number of map (reduce) attempts that are running or ready to run at time t; also called the number of *runnable* map (reduce) attempts. A runnable map (reduce) attempt will actually run only when it is assigned a map (reduce) slot. The number of runnable attempts for a logical map or reduce *task* can be greater than one if *speculative execution* is enabled. In this case, Hadoop will identify *straggler* attempts for a task that are running more slowly compared to other attempts from the same job; and create additional attempts that can be run to complete the task.

Alice can see from Figures 1(a) and 1(b) that, at most points of time, the demand for slots far exceeds the total number of available slots. This observation means that the multi-tenant cluster is almost always under contention, so proper management of resources is essential. The first question that Alice usually has is whether the slots available in the cluster are all being used as much as possible.

Figure 1(c) plots the actual *usage* of map (reduce) slots for the same time period from July 5 to August 4, 2014. Map slot usage $U_m(t)$ is defined at a time t as follows:

$$U_m(t) = \min\{S_m(t) - C_m(t), MAP\_SLOTS\}$$

Here, $S_m(t)$ and $C_m(t)$ denote the number of runnable map attemts and completed map attempts, respectively, at time t. Similarly, $U_r(t)$, $S_r(t)$, and $C_r(t)$ are defined for reduce attempts.

Alice can see from Figure 1(c) that the cluster is almost fully utilized at all points of time. Most cluster operators would take Figure 1(c) to mean that the cluster is performing fine. However, we will dig deeper to get a sense of how much useful work is getting done.

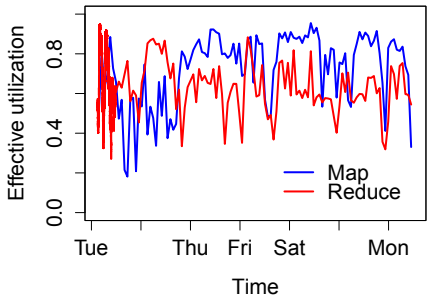

Figure 2: Effective map and reduce utilization

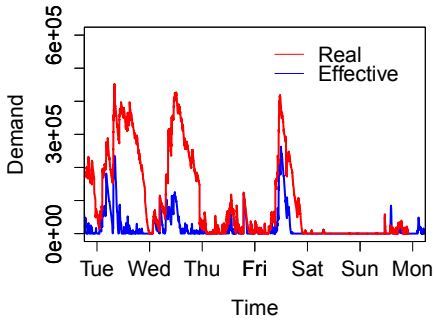

Figure 3: Map demand of the BI pool

Table 1: Statistics of job and attempt status for a one-week period

Type	Percentage (%)
Failed jobs	3
Killed jobs	3.11
Succeeded jobs	93.89
Failed attempts	0.32
Killed attempts	18
Successful attempts	81.68

Let us consider the *effective cluster utilization*, which computes the fraction of cluster usage that went into running attempts that were successful and the job that they were part of was also successful.

We can define the effective utilization $E(t)$ of the cluster as the ratio:

$$E_m(t) = \frac{R_m(t)}{U_m(t)}$$

where $R_m(t)$ is the number of running map attempts at time t that eventually finished successfully and were also part of a job that eventually finished successfully. $E_r(t)$ is defined similarly.

Figure 2 shows $E_m(t)$ and $E_r(t)$ over the week from July 29 to August 4. Alice would be surprised by the results in Figure 2: the average effective utilization for map and reduce is only 72.5% and 67.3%, respectively, over this period. Thus, while the cluster demand and usage look very high, approximately 30% of the cluster resources are being wasted from Alice's perspective. She will want to investigate the reasons that are responsible for the unexpectedly low effective utilization. As part of a diagnosis exercise, Alice can generate statistics as shown in Table 1 for all the MapReduce jobs run over a one-week period.

The table shows that 18.32% of the map and reduce attempts were unsuccessful overall. Several factors can lead to unsuccessful attempts. First, an attempt will be killed if it is a speculative attempt and the original attempt for the task has succeeded; or vice versa. Second, a MapReduce job can fail due to any number of reasons. Common factors include some exception in user code such as a required file not being found.

Third, a MapReduce job can be killed by the submitting user or by Alice. A common case seen in practice is where the job processes heavily-skewed data. Thus, one task of the job gets hit with a massive amount of data that takes a very long time to process. Attempts for this task end up causing heavy load on the machine where they run, prompting Alice to kill the job. A fourth factor is where an attempt gets *preempted*, which we will discuss in Section 2.3.

Figure 4: Interactions across pools

Figure 3 shows the map demand in the BI pool during the July 29 to August 4 time frame. The "Real" plot shows the contributions of this pool to the actual map demand, while the "Effective" plot shows the contributions to the effective utilization. Notice the large gap between the "Real" and "Effective" plots. Thus, from Figure 3, Alice can conclude that the BI pool was one of the contributors to the low effective utilization seen in the cluster. Applications in the BI pool tend to be ad-hoc queries over very large datasets, and submitted by users who are not expert users of Hadoop or SQL.

2.2 Interactions Across Pools

Different applications in a pool contend for the same set of resources. However, can applications in one pool affect those in an-

Table 2: Configuration parameters for BI pool

Name	Setting
min_share_timeout	2 hours
fair_share_timeout	15 minutes
weight	2.0
map_min_share	2287 slots
reduce_min_share	1372 slots
sched_mode	FAIR

other pool? Figure 4 shows four performance metrics for the pools at Rocket Fuel for a week-long interval. In each case, the blue, purple, and green areas represents the BI, ETL2, and APP2 pools respectively. The following observations can be made from Figure 4 as we go through the four performance metrics from top to bottom:

- During July 6 and 7, the BI and APP2 pools had a slight increase in "Jobs Submitted", which is the number of jobs being submitted.
- During the same dates, the BI pool had a more noticeable increase in "Total Cluster Time", which is the amount of time that applications in each pool spend on the cluster.
- The last two performance metrics shown—namely, "Slot Kill Time" and "Slot Fail Time"—denote the amount of time consumed by attempts that are eventually unsuccessful. Note how these numbers went up for the BI and APP2 pools. But much more noticeable is how these times went up for the ETL2 pool.

Figure 4 illustrates the importance of answering one common type of question that Alice has to answer: *What will be the impact of an α% increase in the workload in the Pool X?* This question evolves into a number of other questions because the workload can increase in different ways. For example, natural growth of data sizes can lead to more attempts or to more running time per attempt. Or, creation of a new BI application can lead to an increase in the number of jobs being submitted to the BI pool. Avoiding these questions or providing cursory answers can have catastrophic consequences on cluster performance.

2.3 Controlling the Resource Allocation

Table 2 shows the configuration parameters per pool that control how Hadoop's Fair scheduler allocates resources across different pools. The "min share" and "weight" parameters control how the policy of fair sharing is applied by the Fair scheduler. Choosing the settings for these parameters that provide the best overall workload performance is a challenging task that Alice faces. The task becomes even more challenging while forecasting growth in the demand for different pools, and planning for the resources in advance.

Because of space constraints, we will give only one example, namely, regarding the impact of preemptions done by the Fair scheduler. Premption occurs when the number of slots allocated to a pool goes below its *minimum share* or half its *fair share* for a certain time interval. During preemption, the Fair scheduler will kill the most recently launched attempts in pools that have more attempts running than their fair shares. The work done by the prempted attempts will be lost when they are killed.

The per-pool "timeout" parameters shown in Table 2 determine the time interval mentioned above for which a pool will wait before preemptions start. Alice usually finds it hard to determine whether to enable preemption and, if so, what timeout values to set. On one hand, turning on preemptions is beneficial for performance-critical pools to meet SLAs. However, preemption leads to wasted work. The best setting depends on SLAs, the workload, and cluster capacity.

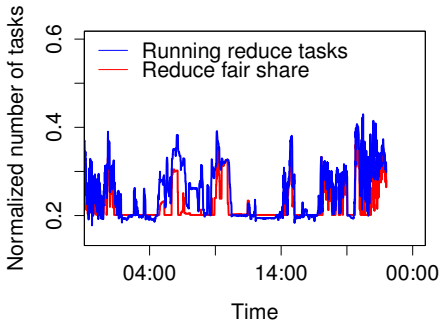

Figure 5: Consequences of long-running reduce attempts in the ETL2 pool

To illustrate the challenges here, Figure 5 shows the reduce fair share and number of running reduce attempts in the ETL2 pool over one day. As can be seen, the ETL2 pool uses more reduce slots than its reduce fair share over a five-hour interval. This trend will be surprising to Alice because preemption of attempts is indeed enabled at Rocket Fuel. The reason for the trend seen in Figure 5 is a combination of the fact that the ETL2 pool has very long-running reduce tasks and the fact that attempts are preempted in increasing order of their running time so far. Given the characteristics of the workload seen in this pool, not enabling preemption could impact severely the chances of meeting SLAs in other pools.

2.4 Tuning the Workload in One or More Pools

Adjusting the per-pool configurations like the ones in Table 2 and Section 2.3 will enable Alice to control the multi-tenancy. However, she would still be playing a *zero-sum game* because giving more resources to one pool means taking away resources from another pool in a contended cluster. In this section, we point out five different ways in which Alice can make it a non-zero-sum game by changing the workload in one or more pools.

Application tuning: Tuning queries or jobs in a pool can improve the resource usage within the pool as well as free up resources for other pools.

Data layout tuning: Tuning data layouts—e.g., changing a table from a text-based format to an encoded columnar format—can change the workload that hits multiple pools. It can be very hard for Alice to predict the impact of such tuning on the overall multi-tenant cluster.

Dependency awareness: Analytics workloads can have a number of data-related dependencies. For example, the input of one job may come from another job in a workflow that contains both jobs. Such dependencies can have significant impact on effective utilization. Consider a MapReduce job that has long-running map attempts. If reduce attempts for this job are started before all map attempts finish, then these reduce attempts may not do any useful work for long periods; while holding on to slots all the while. We have seen significant benefits at Rocket Fuel through tuning that accounts for this behavior.

Workload shifting: As we can see from Figures 1(a) and 1(b), the resource demand on the cluster has highs and lows. Some of the workloads—e.g., those coming from batch applications such as report generation—can be moved to intervals with low resource demand.

Admission control: Admission control can limit the concurrency level of a pool. However, it can be nontrivial for Alice to come up with good admission control policies given different workload characteristics per pool as well interactions among pools. We have experienced cases where changing the admission control policies for one pool led to significant changes in the performance of another pool.

3. OUR COLOSSAL APPROACH

As we illustrated in Section 2, the cluster operations team has to answer a large number of questions in order to manage multi-tenancy. Broadly, we can categorize these questions into three categories.

- *What-is questions:* These are questions aimed at gaining a deeper understanding of multi-tenancy in the cluster. Examples that we saw in Section 2 include understanding the effective utilization of the overall cluster as well as the effective utilization of different pools.
- *What-if questions:* These are questions aimed at understanding the impact of various changes. Examples include understanding the impact of: (a) an increase in a pool's workload, (b) a change in a pool-level configuration parameter such as the preemption timeout, and (c) a change in the layout of a table.
- *What-should questions:* These are questions aimed at identifying the best configurations for tunable parameters given constraints and/or an optimization objective. An example is to find the best setting of pool-level configuration parameters for all pools in order to satisfy constraints (SLAs) on average completion time in some pools, while maximizing the overall cluster utilization.

Colossal is a platform that we are developing to enable cluster operators to get answers to such questions easily. In this section, we will describe the five main components of Colossal and our current implementation of each component. Each component is designed to be pluggable and can have different implementations.

3.1 Extractors

An *Extractor*'s goal is to generate a workload model from raw traces of workload execution. The main reason for generating workload models is to enable anwering What-if questions of the form: what is the impact of a 5% increase in the workload of the BI pool? Workload models enable concrete descriptions of what a 5% increase in a particular workload means.

We have implemented an Extractor for the workload seen on Rocket Fuel's multi-tenant cluster. This Extractor takes as input all job history logs generated by Hadoop for some period of time. From a statistical analysis of the workload, we observed that the job arrival follows a Poisson process with a relatively constant arrival rate for each pool. In addition, the map and reduce attempt duration follows a lognormal distribution (which is similar to the observations in [9]). The task count of a job can also be approximated using a lognormal distribution for each pool. Thus, the Extractor uses maximum likelihood estimation of parameters to fit the observed distributions. Table 3 shows the best-fit workload model for the BI pool.

3.2 Transformers

A *Transformer*'s goal is to generate an actual specification of a workload that will be input to the *Predictor*. We have implemented two different Transformers so far. A simple Transformer takes the raw Hadoop job history logs and transforms it to the workload input specification supported by the Predictor. The more complex Transformer that we have implemented can take the workload model learned by the Extractor from the previous section, and generate various scaled versions of the workload along different dimensions. For example, this Transformer can generate a workload with a 5% increase in the number of map tasks per job in order to represent a 5% increase in overall data volume.

In future, we will be supporting more sophisticated Transformers to connect Colossal with other tuning tools in order to answer questions of the form: What will be the overall impact of changing the data layout of a table T? To answer this question, a data-layout tuning tool (external to Colossal) will have to provide Colossal's Predictor with the new workload that will result from changing the data layout of table T. In this way, Colossal will be able to apply a divide-and-conquer approach to answer hard questions in multi-tenancy that cluster operators have no way of answering today.

3.3 Predictors

A *Predictor* takes three inputs: (i) a multi-tenant workload, (ii) pools and their configuration, and (iii) cluster resource configuration. The goal of the Predictor is to output the predicted schedule of execution of the workload on the cluster. In addition, the Predictor also outputs standard performance metrics associated with a multi-tenant workload.

The Predictor has to be both fairly accurate as well as highly efficient in performance prediction. As we will see in Section 3.5, *Optimizers* may have to call the Predictor multiple times. We could not use a number of Hadoop simulators that have been developed in the past either because they were inefficient in handling large workloads or were too inaccurate due to assumptions made regarding multi-tenancy [4, 5].

Our current implementation of the Predictor is for a cluster where the multi-tenancy is enforced by the Hadoop Fair scheduler. This Predictor is based on a novel algorithm which simulates the Fair scheduler in accelerated *virtual time*. Given the workload trace for an interval T, the predictive algorithm will require a much shorter run-time $t \ll T$ to produce the predicted results. For example, the Predictor requires less than 4 minutes (a significant fraction of which is I/O time for the large workload file) to generate performance predictions for one week of Rocket Fuel's workload.

The key idea used in the predictive algorithm is to map events—such as preemptions, job starts, and job completions—on to a virtual timeline. A finite state machine is then established to process the events in the increasing order of virtual time. While processing an event, the state machine may also alter later events in the virtual timeline, but not the earlier ones. Thus, the running time of the algorithm is proportional to the number of tasks, jobs, and preemptions. This approach has the advantage over most previous approaches in which the running time of simulation also depends on the running time of jobs [4, 10, 5].

3.4 Inspectors

Once the Predictor or Optimizer has generated its output, an *Inspector*'s role is to help the operations team get the answer to their question from the output. In many cases, the overall understanding of the operations team can be further improved by visualizing the time series of metrics in the output. Our current implementation of Inspectors provides visualizations for several built-in performance metrics such as effective utilization as described in the previous section, average job latency for each pool, and others.

3.5 Optimizers

The role of an *Optimizer* in the Colossal platform is to find configuration parameter settings that optimize user-specified performance metrics in a multi-tenant cluster. The optimization is done

Table 3: Workload model of analyst pool (Time is in seconds)

Parameter	Description	Estimated Value
job_arrival_rate	Number of job arrivals per second	0.00282392166
logmean_maps_per_job	Mean of the logarithm of job map count	4.860108
logmean_reduces_per_job	Mean of the logarithm of job reduce count	1.720507
logsd_maps_per_job	Standard deviation of the logarithm of job map count	3.202215
logsd_reduces_per_job	Standard deviation of the logarithm of job reduce count	2.540193
logmean_map_duration	Mean of the logarithm of job task duration	3.765196
logmean_reduce_duration	Mean of the logarithm of reduce task duration	5.154496
logsd_map_duration	Standard deviation of the logarithm of map task duration	0.8837955
logsd_reduce_duration	Standard deviation of the logarithm of reduce task duration	2.043326

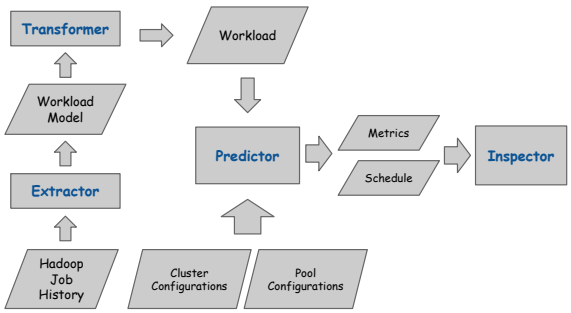

Figure 6: Overview of the Colossal Platform

by searching over the possible parameter space, and calling into the Predictor in order to estimate performance for selected configuration parameter settings. The search process is made efficient by parallelizing using MapReduce as well as using heuristics to prune the search space dynamically.

Specifically, a user can define an objective function *F(M,S)* where M and S are, respectively, the output metrics and schedule from the Predictor. Typical objective functions use three of the built-in metrics: (a) *AvgJobLatency*, which represents the average job latency of each pool, (b) *Throughput*, which represents the throughput (number of jobs per second) of each pool, and (c) *EffectiveUtil*, which represents the effective utilization defined in Section 2. Additionally, users specify the parameters (i.e., the free variables in the search) for which they want recommended settings. For example, any subset of the configuration parameters listed in Table 2 can be specified as free variables.

4. DISCUSSION AND RELATED WORK

Efficient processing of big data has given rise to multi-tenant, big data clusters where multiple applications run on and share the same resources and data. Such multi-tenancy poses significant challenges for human operators of the cluster to tune the cluster to meet performance requirements that often conflict each other. The research and open-source communities are doing a lot of work in this area. The main threads of work include: (a) building cluster operating systems like Mesos [6] and YARN [11] that have the resource provisioning and isolation mechanisms to support multi-tenancy; (b) algorithms and policies to allocate resources in ways that are efficient as well as fair to all tenants [8, 2, 7]; and (c) ability to support multiple tenants on as few software and hardware resources as possible (e.g., [1]).

In this paper, we observed that, in practice, the responsibility of managing multi-tenancy falls largely on the operations team that manages a multi-tenant cluster. We used operational experiences from a large, multi-tenant cluster to discuss the challenges that arise. Specifically, we observed that cluster operators have few effective tools to make data-driven decisions while managing multi-tenancy. We described Colossal, a platform that we have developed to address this gap.

Space contraints precluded us from presenting results from an experimental evaluation. Overall, the primary advantages that Colossal offers are:

- *Efficiency*: Colossal can predict, with acceptable levels of accuracy, the performance metrics and schedule of one week's workload at Rocket Fuel (approximately, 55,000+ MapReduce jobs and 36 million attempts) in under four minutes on a commodity machine.
- *Flexibility*: Colossal allows for easy abstraction, modification, and prediction of the workload over any specified time period lengths.
- *Modularity*: The components that generate the workload as well as predicted performance metrics are pluggable. Thus, Colossal can be customized based on the characteristics and needs of different multi-tenant environments.

5. REFERENCES

[1] S. Das, V. R. Narasayya, F. Li, and M. Syamala. Cpu sharing techniques for performance isolation in multitenant relational database-as-a-service. *PVLDB*, 7(1):37–48, 2013.

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

[3] Hadoop Fair Scheduler. hadoop.apache.org/docs/r1.2.1/fair_scheduler.html.

[4] Hadoop SLS Simulator. hadoop.apache.org/docs/r2.3.0/hadoop-sls/SchedulerLoadSimulator.html.

[5] S. Hammoud, M. Li, Y. Liu, N. K. Alham, and Z. Liu. Mrsim: A discrete event based mapreduce simulator. In *FSKD*, pages 2993–2997. IEEE, 2010.

[6] 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*, 2011.

[7] 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.

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

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

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

[11] Apache Hadoop NextGen MapReduce (YARN). http://hadoop.apache.org/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/YARN.html.