Chameleon: The Performance Tuning Tool for MapReduce Query Processing Systems

Edson Ramiro Lucsa Filho[1], **Ivan Luiz Picoli**[2], **Eduardo Cunha de Almeida**[2], **Yves Le Traon**[1]

[1] University of Luxembourg

{edson.lucas,yves.letraon}@uni.lu

[2]Federal University of Paraná

{ilpicoli,eduardo}@inf.ufpr.br

Abstract. *Chameleon is a tuning advisor to support performance tuning decision-making of MapReduce administrators and users. In MapReduce query processing, a query is translated into a set of jobs, i.e., query plan. For administrators, Chameleon can be a powerful tool for observing query plan workloads and their impact in large-cluster machine setups in terms of computing resource consumptions. For users, Chameleon provides a set of functionalities to tune query plans and observe performance improvements while testing different tuning knobs. Chameleon embeds a tuning mechanism based on a hash index to map jobs to tuning knobs. The hash index allows users defining specific tuning knobs for jobs, while a clustering algorithm is responsible for finding jobs with similar resource consumptions to receive the same tuning. In this demonstration we outline the functionalities of Chameleon and allow users interacting with it by sending and tuning queries for auditing performance improvements.*

1. Introduction

In the last years, we have seen the rise of MapReduce Query Processing Systems that translate SQL-like queries into Map Reduce programs, such as Hive [Ashish Thusoo 2009], DryadLINQ [Isard et al. 2007], Pig [Gates et al. 2009], and Shark [Xin et al. 2013]. The advantage of these systems is that they make declarative programs leveraging the MapReduce benefits. For instance in Hive, every incoming query is translated into a set of jobs to be executed in distributed machines, where a job is a program with a set of query operators, the performance tuning knobs and data input. This set of jobs represents the query execution plan in the form of a Directed Acyclic Graph (DAG).

In these systems, performance tuning knobs are either defined in a central configuration file or included into the job source-code (similar to SQL hints). However, in either ways, the same tuning configuration is propagated to all jobs within the DAG, forbidding users to define specific tuning knobs for each job. The propagation of the same tuning can be treacherous, since distinct query operators and varied input data sizes lead to different usages of computing resources (the *TableScan* operator requires disk bandwidth, the *Sort* operator requires memory). Therefore, performance of MapReduce queries depends on specific tuning configuration for each job. Instead of coarse-grained tuning, at the level of a query, we promote a fine-grained tuning, with specific tuning knobs per job. The

http://www.inf.ufpr.br/erlfilho/chameleon.mp4

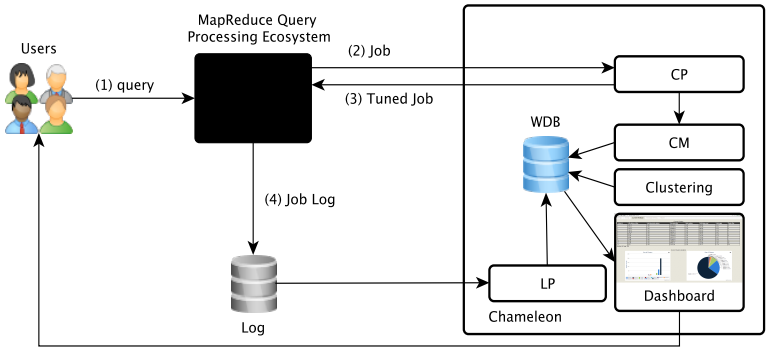

Figure 1. The Chameleon architecture and its integration with the MapReduce Ecosystem.

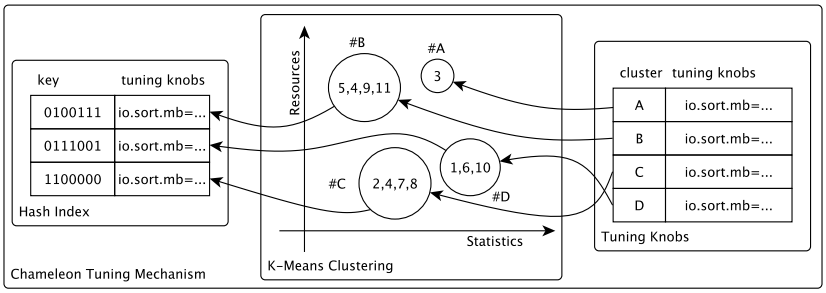

Figure 2. K-means updating the tuning knobs in the Chameleon Hash Index.

issue we address in this paper is the tool-environment for materializing such fine-grained tuning.

Chameleon is a tuning advisor to MapReduce Query Processing Systems. Its main goal is supporting MapReduce users to make performance tuning decisions. Chameleon uses a tuning mechanism based on a hash index. The index key is a canonical representation of a job in the form of a bitmap. The index values represent the tuning knobs. Chameleon implements the K-means unsupervised machine learning algorithm to cluster jobs with equivalent resource consumptions and update the tuning knobs. Clusters are computed from the resource consumption statistics of the execution log files. Once a new job gets its key, it is automatically mapped to a tuning setup. Chameleon does not calculate the actual tuning, rather than, it reuses precomputed tuning (from users or third party MapReduce tuning systems) and optimize queries upfront the execution. Indeed, Chameleon allows ad-hoc jobs receiving proper tuning without monitoring or sampling executions. In this demonstration we outline the functionalities of Chameleon and allow users interacting with it by sending and tuning queries for auditing performance improvements.

2. Architecture

Figure 1 illustrates the Chameleon architecture interacting with the MapReduce Query Processing Ecosystem. The architecture has six modules, described as follows:

- **Code Parser** (CP) is responsible for reading the source-code of each job and creating a canonical representation for it (i.e., the hash index key). Initially, Chameleon

Figure 3. Dashboard: The history list of executed queries.

assumes that the resource consumptions of different jobs can be said equivalent if they implement equivalent query operators. Then, a key maps jobs to specific tuning knobs. After clustering, the keys will be used to update the tuning knobs. Let τ be the query operators and the domain of τ be the list of operators $\{op_1, \ldots, op_m\}$. A key κ is a one-to-one mapping $(M : \tau \rightarrow \{< b_1, \ldots, b_{|\tau|} > | b_i \in \{0, 1\}, i = 1, \ldots, |\tau|\})$. The key is represented as a bitmap, called κ, with a bit turned on or off accordingly to the existence of the operator in the source code. For instance, let us say that a job j_0 implements only the *TableScan* operator, than, $\tau_0 = \{TableScan\}$ and, for instance, $|\tau| = 5$. The calculated key for this job is $\kappa_0 = \{00001\}$, where the bit for the $TableScan$ operator has been switched to 1 in τ_0. The number of query operators supported by the index is represented by τ. This number can be incremented as news programs are parsed. The query systems are evolving and new operators can be mapped without additional definitions, adding a new operator to the key and incrementing $|\tau|$.

- The **Clustering** module performs the K-means algorithm over the execution statistics gathered at the log files. Clustering is the foundation of the Chameleon tuning. K-means creates clusters for jobs with similar usage of computing resources. Once jobs in the same cluster present similar resource consumptions, they are allowed to receive the same tuning setup. Users may interact with K-means by setting 1 to 65 computing resources traced in the log files used. In addition, users can configure the execution of the K-means algorithm to update clustering information. The *refresh_time* configuration refers to the periodicity to perform K-means. The *number_of_classes* configuration refers to the number of clusters created. The *clustering_execs* configuration defines how many times the K-means algorithm is performed within the *refresh_time*.
- **Code Mapper** (CM) maps keys to clusters. For this, Chameleon computes the occurrences of keys per cluster. The algorithm is straightforward, the key maps to the cluster in which it appears the most. Figure 2 illustrates the keys matching the tuning knobs in the hash index. For instance, lets say jobs 1, 3 and 6 share the same key $\{0111001\}$, but in different clusters (#A and #D). Their key will map to cluster #D with two occurrences. This means, key $\{0111001\}$ will receive the tuning knobs from cluster #D. The goal is allowing upcoming jobs reusing precomputed tuning setup. Once a new job gets its key, it is mapped to a cluster and automatically receives the tuning setup from such cluster. In this way, Chameleon can reuse precomputed tuning (from users or third party MapReduce tuning systems) and optimize queries upfront the execution.
- **Log Parser** (LP) periodically reads the query log files traced by the MapReduce Query Processing Ecosystem in order to extract execution statistics; for instance,

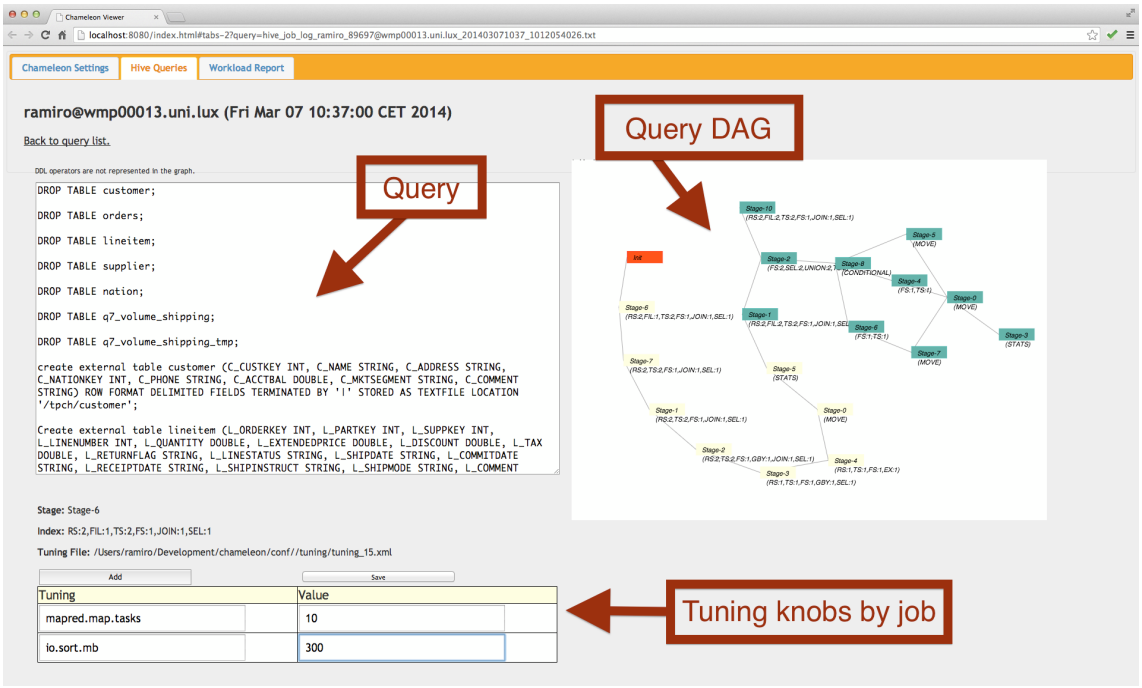

Figure 4. Dashboard: Query analysis interface.

Tuning Knobs	Range of Values
io.sort.mb	70-250
io.sort.factor	7-10
io.sort.record.percent	0.05-0.5
io.sort.spill.percent	0.80-0.99
io.file.buffer.size	4096-131072
mapred.child.java.opts	Xmx512m - Xmx1g Xms256m - Xms512m
mapred.job.shuffle.merge.percent	0.66-0.99
mapred.reduce.parallel.copies	5-10

Table 1. List of the Hadoop tuning knobs.

the number of bytes that was processed at the map and reduce phases.

- **Workload Database** (WDB) is the internal information database that hosts the hash index and tuning knobs computed by third party as depicted by Table 1. It also hosts the information required to perform K-means, such as: the *statistics* from the log files and the clusters.
- The **Dashboard** module provides the Chameleon web interface with three tabs. The *Chameleon Settings* tab is reserved to setup the internal configurations, such as: database connection, path to the MapReduce Query Processing Ecosystem, and K-means setup. The *Queries* tab, illustrated in Figure 3, summarizes the execution history. After selecting one of the queries for tuning, as we illustrate in Figure 4, the *Dashboard* module presents the query plan with its internal jobs and source-codes, and the tuning knobs applied. As illustrated in Figure 5, the *Workload Report* summarizes the workload characteristics of each cluster. Once

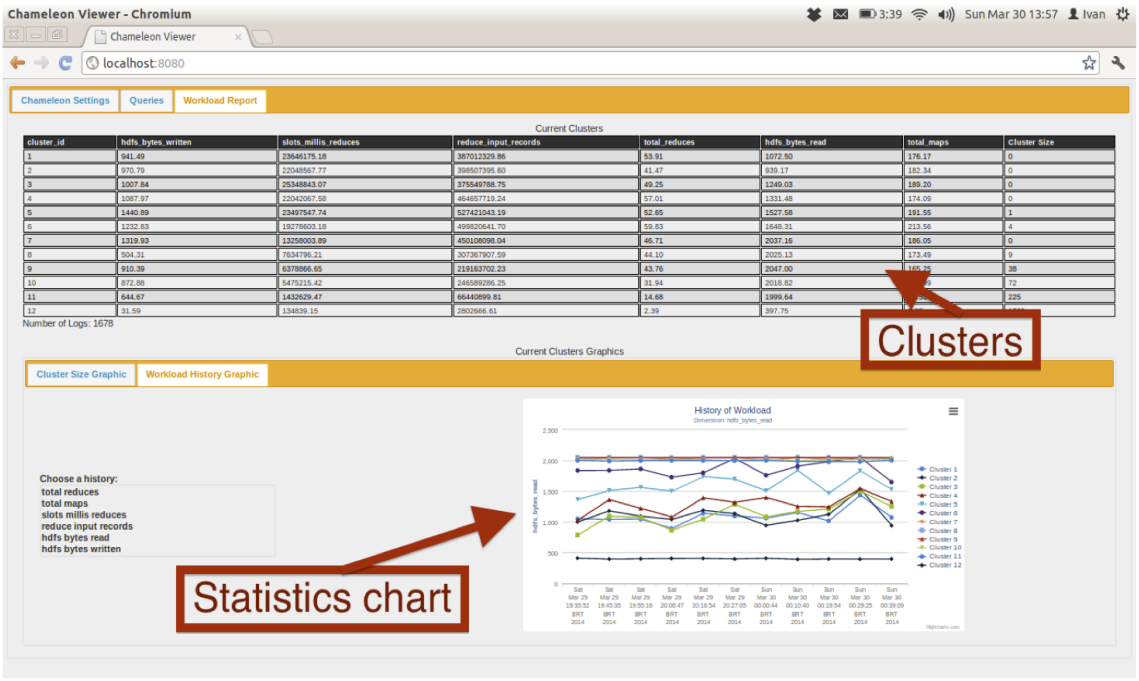

Figure 5. Dashboard: The workload analysis interface.

the users know the workload characteristics, they are able to define specific tuning parameters for each cluster.

3. Demonstration overview

We will present the Chameleon functionalities, allowing users to interact with it by sending queries via HiveQL command line interface and setting different tuning setups to observe performance improvements. Users will also interact with Chameleon via web interface to analyze the execution workloads. The current implementation of Chameleon supports the Apache Hive 0.11.0 and Apache Hadoop 1.2.1. The demonstration agenda is organized, as follows:

1. **Start Chameleon Internal Modules** We will launch the Chameleon modules that will be up and running in background during the demonstration. We will present a brief description of the Chameleon configuration files, including the K-means setup.
2. **Query submission**: The Apache Hive will have pre-loaded data of the TPC-H database. We will provide the 22 TPC-H queries written in HiveQL, but users will be able to submit any query via Hive Command Line Interface to see the functionalities of Chameleon. Users will observe Chameleon tuning each job generated by queries followed by the proper execution.
3. **Dashboard**: Users will be able to access the Dashboard and interact with the *Queries* tab to see the execution history of jobs. They will be able to interact with any submitted query by checking the query source-code, the query DAG with its internal jobs and the tuning knobs applied to each job. At this point, users will be able to interact with the *Workload Report* tab to analyze the charts and tables that describe the clusters and their workload.

Figure 5 depicts the clusters and their execution statistics. Users will visualize a combination of statistics used by the K-means algorithm and their respective values. The header plots the *statistics* chosen by the user in the Chameleon settings, where each column represents one of the *statistics*. The column values are the average *statistic* values computed by K-means. The last column represents how many jobs have been clustered. Users will visualize the *Statistics* chart, which drills down to the execution statistics of each cluster. This chart is a tool to check the ongoing workload in the MapReduce Query Processing Ecosystem. For instance, the number of bytes read and written by the HDFS.

4. **Tuning Query**: Users will be invited to redefine the tuning parameters for each cluster or import them from third party tuning systems. Then, users will be able to re-execute the same query via Hive Command Line Interface in order to observe performance improvements.

4. The Take-Away Message

Chameleon is a tuning advisor to support performance tuning decision-making of MapReduce users and administrators. For users, Chameleon provides a set of functionalities to tune query plans and observe performance improvements while testing different tuning knobs. Chameleon does not calculate the actual tuning, rather than, it reuses precomputed tuning (from users or third party MapReduce tuning systems) and performs intra-query optimization upfront the execution. It also enables jobs from different queries sharing the same tuning. For administrators, Chameleon can be a powerful tool for observing query workloads and their impact in large-cluster machine setups in terms of computing resource consumptions.

References

Ashish Thusoo, J. S. S. (2009). Hive- A Warehousing Solution Over a Map-Reduce Framework. *Proceedings of the VLDB Endowment*.

Gates, A. F., Natkovich, O., Chopra, S., Kamath, P., Narayanamurthy, S. M., Olston, C., Reed, B., Srinivasan, S., and Srivastava, U. (2009). Building a high-level dataflow system on top of Map-Reduce: the Pig experience. *Proceedings of the VLDB Endowment*, 2(2):1414–1425.

Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007). Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review*, 41(3):59.

Xin, R. S., Rosen, J., Zaharia, M., Franklin, M. J., Shenker, S., and Stoica, I. (2013). Shark: SQL and rich analytics at scale. In *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, page 13, New York, New York, USA. ACM Press.