HAL
archives-ouvertes.fr

Intra-query Adaptivity for MapReduce Query Processing Systems

Edson Ramiro Lucas Filho, Eduardo Cunha de Almeida, Yves Le Traon

▶ **To cite this version:**

Edson Ramiro Lucas Filho, Eduardo Cunha de Almeida, Yves Le Traon. Intra-query Adaptivity for MapReduce Query Processing Systems. IDEAS 2014 : 18th International Database Engineering & Applications Symposium, Jul 2014, Porto, Portugal. 2014. <hal-01018087>

HAL Id: hal-01018087

https://hal.inria.fr/hal-01018087

Submitted on 3 Jul 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Intra-query Adaptivity for MapReduce Query Processing Systems

Edson Ramiro Lucas Filho[1], Eduardo Cunha de Almeida[1,2], and Yves Le Traon[1]

[1]University of Luxembourg – {edson.lucas, yves.letraon}@uni.lu

[2]Federal University of Paraná, Brazil – eduardo@inf.ufpr.br

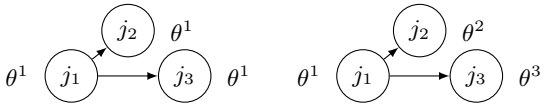
ABSTRACT

MapReduce query processing systems translate a query statement into a query plan, consisting of a set of MapReduce jobs to be executed in distributed machines. During query translation, these query systems uniformly allocate computing resources to each job by delegating the same tuning to the entire query plan. However, jobs may implement their own collection of operators, which lead to different usage of computing resources. In this paper we propose an adaptive tuning mechanism that enables setting specific resources to each job within a query plan. Our adaptive mechanism relies on a data structure that maps jobs to tuning codes by analyzing source code and log files. This adaptive mechanism allows delegating specific resources to the query plan at runtime as the data structure hosts specific precomputed tuning codes.

1. INTRODUCTION

MapReduce query processing systems, such as Hive [1] and Shark [5], translate SQL-like queries into a set of jobs (i.e., query plan), where each job is a complete MapReduce program consisting of a reference to its input data, its tuning knobs, and a collection of operators (e.g., Join, Filter). During query translation, the MapReduce query processing systems allocate computing resources uniformly to each job by delegating the same tuning for the entire query plan. However, each job within the query plan has a different collection of operators that lead to a different resource usage.

Like relational database systems, performance can be improved by choosing the correct tuning values, which can be set by users into the query source code (similar to SQL hints) or in central configuration files. Setting the suitable values is a difficult task for users due to the number of variables involved [2]. However, the same tuning is propagated across the entire query plan, which uniformly allocates computing resources to each job (i.e., uniform tuning). Figure 1a illustrates a query plan with three jobs j_1, j_2, and j_3 receiving the same tuning θ^1. This uniform tuning approach is carried out by the current tuning techniques [2]. Our objective is to present an adaptive query processing mechanism, where each job receives specific tuning (see Figure 1b).



(a) Uniform tuning (b) Adaptive tuning

Figure 1: MapReduce query tuning mechanism.

2. ADAPTIVE QUERY PROCESSING

Our hypothesis is that jobs from the same query plan have different usage of computing resources when they implement a different collection of operators. Thus, adaptive tuning must be applied to delegate specific computing resources to each job. To illustrate the load fluctuation, Figure 2 depicts the disk consumption of the jobs from the TPC-H [1] query #16. In this paper we propose an adaptive query mechanism in order to set specific resources to each job. Our mechanism is defined in three main components: (1) Instantaneous Tuner, (2) Workload Monitor, and (3) Tuning Discoverer.

2.1 Instantaneous Tuner

The *Instantaneous tuner* is a component that enables intra-query adaptivity by setting specific tuning for each job of a query plan. It is based on a hash index where the key κ^i is a representation of a job in the form of a bitmap, and the value is the set of tuning knobs represented by θ^i. Once a job gets a key, it is automatically mapped to the related tuning knobs. We assume that different jobs have similar resource consumption if they implement equivalent collection of operators. To allow comparison for equivalence, the *Instantaneous tuner* reads the source-code of each job in order to compute its hash key. The key is represented as a bitmap κ with a bit switched on if the operator is used by the job, or off otherwise. Let τ be the query operators and the domain of τ be $\{op_1, \ldots, op_m\}$. A key κ is a one-to-one mapping

[1]www.tpc.org

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IDEAS '14, July 07 - 09 2014, Porto, Portugal

Copyright is held by the owner/authors. Publication rights licensed to ACM.

ACM 978-1-4503-2627-8/14/07 ...$15.00

http://dx.doi.org/10.1145/2628194.2628202.

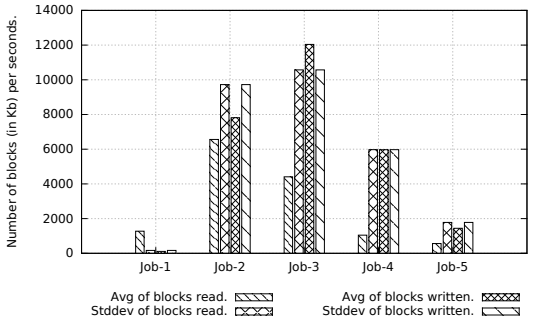


Figure 2: Disk consumption per job from TPC-H query 16.

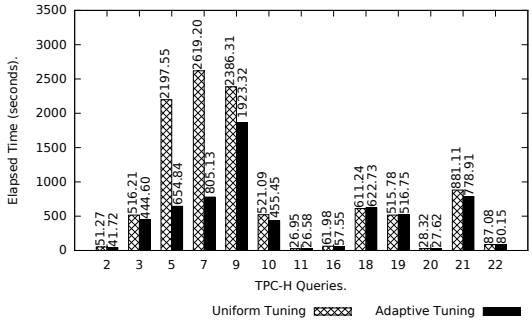


Figure 3: Execution time of TPC-H queries on Hive with uniform and adaptive tuning methods.

$(M : \tau \rightarrow \{< b_1, \ldots, b_{|\tau|} > | b_i \in \{0,1\}, i = 1, \ldots, |\tau|\})$. Suppose that the job j_0 implements only $\tau_0 = \{Join\}$ and $|\tau| = 5$. A possible key is $\kappa_0 = \{00001\}$ with the bit for the *Join* operator switched on. The size of $|\tau|$ represents the number of query operators kept by our hash index and increments as the programs are parsed. The size of $|\tau|$ varies because the API of the query systems are evolving and new operators can be automatically mapped without further definitions. It suffices to append 0 to each key to map a new operator to the hash index and increment $|\tau|$. Future work includes enhancing our adaptive tuning mechanism by computing the bitmap key for jobs from different query languages, and hosting tuning knobs for different databases.

2.2 Workload Monitoring

The *Workload Monitoring* component traces the resource consumption of the jobs. The tracing information is later used together with the execution log files from the MapReduce environment (e.g., Hadoop) to feed the unsupervised clustering algorithm. The algorithm generates clusters of jobs with similar resource consumptions to allow applying the same tuning θ^i. To map keys to clusters, our tuning mechanism computes the occurrences of keys per cluster. The algorithm directly maps keys to the cluster in which they appear the most. The objective is reusing precomputed tuning setup in the upcoming jobs. 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 the tuning mechanism can reuse precomputed tuning and optimize queries upfront the execution. Future work includes evaluating different machine learning algorithms in order to improve clustering.

2.3 Tuning Discoverer

The *Tuning Discoverer* component is responsible for generating the appropriate tuning setup for each cluster created by the *Workload Monitor*. The current approaches use heuristics [3] and simulation [4] to find the best tuning setups. However, these techniques add a considerable overhead to the entire process and are not properly suitable for ad-hoc queries, since they need to execute or simulate the query at least once. Future work includes the generation of optimal tuning values. For this generation we plan to explore linear regression and heuristics.

3. EVALUATION

Through a series of comprehensive experimentations on the TPC-H Benchmark for Hive, we demonstrate that the tuning approach of the popular Apache Hive query system is inefficient as it wastes computing resources and increases query response time. For the uniform tuning, the tuning knobs were defined based on the *rules-of-thumbs* in the central configuration files of Hive and Hadoop. For the adaptive tuning we intercepted each job before executing and inserted new tuning knobs based on the *rules-of-thumbs*, executing each job with specific tuning. During query execution we collected CPU, memory, network, and disk information using the SysStat[2] tool. Figure 3 depicts 13 queries that the adaptive tuning mechanism decreased the response time. The adaptive tuning mechanism differs from existing ones by not creating job profiles or simulating the execution, which is convenient for ad-hoc queries as showed by the experiments.

Acknowledgement: Supported by the National Research Fund, Luxembourg - TOOM Project: C12/IS/4011170 and SERPRO/UFPR.

4. REFERENCES

[1] J. S. S. Ashish Thusoo. Hive- A Warehousing Solution Over a Map-Reduce Framework. *VLDB Endowment*, pages 1626–1629, 2009.

[2] H. Herodotou, H. Lim, G. Luo, N. Borisov, and L. Dong. Starfish : A Self-tuning System for Big Data Analytics. *CIDR '11*, pages 261–272, 2011.

[3] G. Liao, K. Datta, T. L. Willke, V. Kalavri, V. Vlassov, and P. Brand. Gunther: search-based auto-tuning of mapreduce. volume 8097 of *Euro-Par'13*, pages 406–419, Aug. 2013.

[4] G. Wang, A. R. Butt, P. Pandey, and K. Gupta. Using realistic simulation for performance analysis of mapreduce setups. In *LSAP '09*, page 19. ACM Press, June 2009.

[5] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: SQL and rich analytics at scale. In *SIGMOD '13*, page 13. ACM Press, June 2013.

[2]https://github.com/sysstat/sysstat