The Uniform Tuning Problem on SQL-On-Hadoop Query Processing

Edson Ramiro Lucas Filho
Federal University of Parana, Brazil
Supervised by: Eduardo Cunha de Almeida
Expected Graduation: Oct 2018.
erlfilho@inf.ufpr.br

ABSTRACT

SQL-On-Hadoop systems translate a given query into several MapReduce jobs. Each job executes a different set of query operators over different input data sets, which leads to distinct resource consumption patterns. Once each job has a different resource consumption pattern they should receive tailor made tuning setup. However, SQL-On-Hadoop systems propagate the same tuning to every job in the query plan because they are not able to apply a specific tuning setup per job. Propagating the same tuning through the query plan is a problem because it drives the query to suboptimal performance and drives tuning advisors to re-profile similar jobs several times. In our research we characterize this problem and propose a solution. Preliminary results show that our approach can reduce the number of profiles required by tuning advisors in 67% for TPC-H.

Keywords

Query Tuning; SQL-On-Hadoop; Directed Acyclic Graph

1. PROBLEM

The MapReduce programming model scales the processing of large amounts of data on distributed machines by decomposing the computation in a divide-and-conquer manner. However, to write MapReduce programs developers need to be acquainted with the details of the MapReduce frameworks (e.g., Spark [12], Hadoop [1]). The complexity of writing such programs is simplified by declarative languages provided by query processing engines built on top of MapReduce frameworks like Pig [8], Hive [10] and SparkSQL [2] (a.k.a., SQL-On-Hadoop systems [3]).

SQL-On-Hadoop systems are designed for batch and analytics processing that demand long term running jobs. When developers need to decrease the execution time of such jobs they can configure the tuning setup to achieve better performance. Once the tuning setup is configured, they are applied in the query source code, where this tuning setup has been generated by tuning advisors or hand made by developers.

SIGMOD'17 Student Research Competition May 14-19 2017, Chicago, IL, USA

ACM ISBN 978-1-4503-4199-8/17/05.

DOI: http://dx.doi.org/10.1145/3055167.3055172

Table 1: The MapReduce tuning advisors.

Tuning System	Hadoop	Improvement
Gunther	1.x.x	25%-33%
MR-COF	0.20.2	35%
MRTuner	1.1.1	1x
Panacea	1.x.x	1.6x-2.9x
MROnline	YARN	30%
JellyFish	YARN	24%-65%
GeneExpr	1.2.1	46%-71%

During the query processing the SQL-On-Hadoop systems translate a given query into a query plan. The query plan consists of a set of MapReduce jobs organized into a Directed Acyclic Graph (DAG). Each job of the query plan has a different set of query operators and a different set of input data, which leads to distinct resource consumption patterns. Thus, once a query plan consists of several jobs with distinct resource consumption patterns the tuning setup should be tailor made for each job in order to achieve optimal performance. However, the SQL-On-Hadoop systems propagate the same tuning to every job because they are not able to differentiate jobs within the query plan in order to apply a specific tuning setup per job. Our hypothesis is that jobs that execute the same query operators over similar input data set have similar resource consumption pattern and, then, should receive the same tuning.

We named the problem of propagating the same tuning through the query plan as the Uniform Tuning Problem (UTP). In our research we identify and characterize its consequences. Our findings were obtained with Hive and Hadoop and indicates that in the presence of the UTP the tuning advisors may generate multiple tuning options not applicable for one query and lead to misconfigurations.

2. RELATED WORK

Table 1 presents recent tuning advisors like Panacea [7], Gunther [5], MR-COF [6], MRTuner [9], MROnline [4], and JellyFish [11] and their improvement rate, as well as the supported Hadoop version. Speedups may vary from 24% up to 2.9x of the overall execution time depending on the workload. However, these tuning advisors are designed for MapReduce frameworks and lack on dealing with the UTP.

The tuning advisors generate multiple tuning options, because they provide one tuning setup per job, but they also leave to the developers the task of choosing the right tuning setup to be applied in the query. Once SQL-On-Hadoop systems can accept only one tuning setup per query this leads

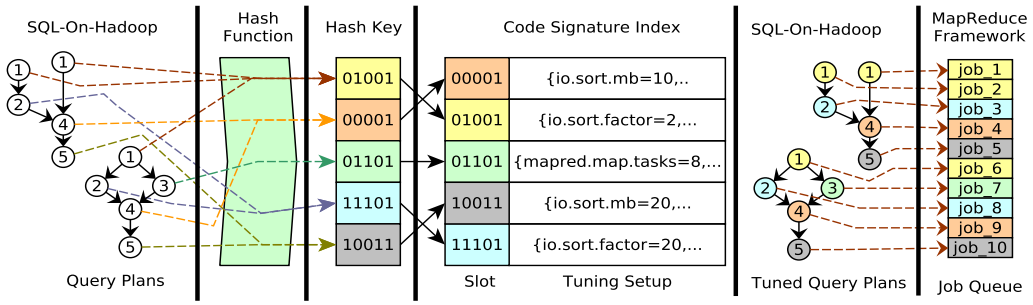

Figure 1: The process of calculating the code signature for each job of the query plan upfront execution.

to the UTP. The multiple tuning options lead developers to misconfigure the query because the chosen tuning is tailor made for a specific job within the query plan, but it will be applied for jobs with different resource consumption.

In order to generate the tuning setup some tuning advisors profile the jobs, others simulate their execution, and others analyze their logs. Despite where tuning advisors profile, simulate or analyze the logs of the jobs they should not use the entire query workload as one workflow to avoid the UTP during the generation of the query setup. Profiling a whole query generates a generic view of the workload and may lead the tuning advisors to draw sub-optimal tuning setup and may lead to waste of resources. Thus, we observe that tuning advisors should be able to profile, simulate or analyze the logs of individual jobs, but with a method to apply the tuning setup to each job of the query plan. Preferably, to apply the tuning setup upfront execution because of the ad-hoc queries inherent to the analytics workloads.

3. APPROACH

Figure 1 illustrates our approach, the Code Signature Index, that applies a specific tuning setup to each job of the query plan. Users submit queries to the SQL-On-Hadoop, which translates the queries into query plans. After the query translation and before the SQL-On-Hadoop system queue jobs in the MapReduce framework we calculate a code signature for each job. The code signature is a hash value built with the order of magnitude of the input data size and the list of query operators of the job. The Algorithm 1 describes the process to calculate the code signature. Other information can be easily added to the hash such as selectivity of the operators.

Our approach implements a hash table where the code signatures are keys and the tuning setups values. Thus, by using the code signature we can map upfront execution each job to a specific tuning. Also, the same code signature repeats among the jobs from the same query plan and among jobs from different query plans.

Our approach links the appropriate tuning setup to the code signature by using third part tuning advisors to generate the tuning setup. When a code signature is added to the hash table, there will be no tuning setup. Thus, at this point, our approach calls a tuning advisor to generate the corresponding tuning setup. From this point, all subsequent jobs with the same code signature will receive the same tuning setup. In order to update the hash table we may execute in background different tuning advisors and then choose the best tuning among the options.

Algorithm 1: How to calculate the code signature.

```
Data: a job of a query plan
Result: code signature of the given job
scientificNotation = convertToScientific(job.inputSize);
exponent = scientificNotation.getExponent;
for op : job.queryOperators do
    if list.has(op) then
        list.add(op, list.get(op)+1);
    else
        list.add(op, 1);
    end
end
return code_signature = exponent:list;
```

4. RESULTS

In this study we observed that MapReduce tuning advisors lack on optimize SQL-On-Hadoop systems because the presence of the Uniform Tuning Problem. Our Code Signature Index approach copes the UTP mapping each job upfront its execution to specific tuning setup. Also, it decreases the number of profile operations required by the tuning advisors, which are very costly.

In order to identify the number of profile operations decreased by our approach we executed the 22 TPC-H queries against our modified version of Hive (v0.13.1) over Hadoop (v2.7.3) on scale factors of 10, 30 and 100. Our experimental setup is a 3 node cluster. Each node has a Intel Core i3, 4GB of RAM and 1TB of disk. The queries on scale factors of 10 and 30 produced 163 jobs and on factor of 100 produced 164 jobs. For the 3 factors we found 55 unique code signature, which means that we do not need to profile 67% of the jobs for TPC-H because they execute the same query operators over the similar input data sets. We are studying the relation of the selectivity and other factors on the code signature. We are also working to support other query processing engines like SparkSQL.

5. ACKNOWLEDGMENTS

This work is funded by the Brazilian Innovation Agency (FINEP) project (LNCC-FACC-CICN, 01.13.0056.00), CNPq (grant 441944/2014-0).

6. REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org, 2014.

[2] M. Armbrust, A. Ghodsi, M. Zaharia, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, and M. J. Franklin. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*, pages 1383–1394, New York, New York, USA, may 2015. ACM Press.

[3] A. Floratou, U. F. Minhas, and F. Özcan. SQL-on-Hadoop: Full Circle Back to Shared-Nothing Database Architectures. *Proceedings of the VLDB Endowment*, 7(12):1295–1306, aug 2014.

[4] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, and N. Fuller. MRONLINE: MapReduce Online Performance Tuning. *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing - HPDC '14*, pages 165–176, 2014.

[5] G. Liao, K. Datta, T. L. Willke, V. Kalavri, V. Vlassov, and P. Brand. Gunther: Search-Based Auto-Tuning of MapReduce. *Proceedings of the 19th International Conference on Parallel and Distributed Computing, Euro-Par*, 8097:406–419, aug 2013.

[6] C. Liu, D. Zeng, H. Yao, C. Hu, X. Yan, Y. Fan, C.-H. Hsu, and L. Yang. MR-COF: A Genetic MapReduce Configuration Optimization Framework. In G. Wang, A. Zomaya, G. Martinez Perez, and K. Li, editors, *Theoretical Computer Science*, volume 6082, pages 338–347–347, Cham, 2015. Springer International Publishing.

[7] J. Liu, N. Ravi, S. Chakradhar, and M. Kandemir. Panacea: Towards Holistic Optimization of MapReduce Applications. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization - CHO '12*, page 33, New York, New York, USA, mar 2012. ACM Press.

[8] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: a Not-So-Foreign Language for Data Processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data - SIGMOD '08*, page 1099, New York, New York, USA, jun 2008. ACM Press.

[9] J. Shi, J. Zou, J. Lu, Z. Cao, S. Li, and C. Wang. MRTuner: A Toolkit to Enable Holistic Optimization for MapReduce Jobs. In *Proceedings of the VLDB Endowment*, volume 7, pages 1319–1330. VLDB Endowment, aug 2014.

[10] 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. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, aug 2009.

[11] Xiaoan Ding, Yi Liu, Depei Qian, X. Ding, Y. Liu, and D. Qian. JellyFish: Online Performance Tuning with Adaptive Configuration and Elastic Container in Hadoop YARN. In *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, volume 2016-Janua, pages 831–836. IEEE, dec 2016.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark : Cluster Computing with Working Sets. *HotCloud'10 Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, page 10, jun 2010.