A Non-Uniform Tuning Method for SQL-on-Hadoop Systems

Edson Ramiro Lucas Filho, Renato Silva de Melo, and Eduardo Cunha de Almeida

Universidade Federal do Paraná, Brazil
{erlfilho,rsmelo,eduardo}@inf.ufpr.br

Abstract. A SQL-on-Hadoop query consists of a workflow of MapReduce jobs with a single point of configuration. This means that the developer tunes hundreds of tuning parameters directly in the query source code (or via terminal interface), but the system assumes the same configuration to every running job. The lurking problem is that the system allocates computing resources uniformly to every running job, even if they present different consumption needs (after all the tuning setup is the same). In this paper, we demonstrate that such uniform allocation of resources drives the query to inefficient performance. We claim that applying a non-uniform allocation method to define a customized tuning setup for each job can outperform the uniform allocation. Ultimately, we demonstrate that searching for specific tuning setup is an optimization problem with polynomial cost.

Keywords: Tuning · Self-Tuning · SQL-on-Hadoop · MapReduce.

1 Introduction

Ever since the proliferation of SQL-on-Hadoop engines [2,7,3,23,8,14,22,1] the number of configuration parameters exposed by such systems, and by the underneath MapReduce systems [9,20], has grown considerably, as presented in Figure 1. For instance, the SQL-on-Hadoop system Apache Hive has increased from 96 configuration parameters in its 0.6.0 release up to 989 parameters in the release 3.0.0. The underlying processing engine Apache Hadoop has increased from 104 configuration parameters in its 0.12.0 release up to 530 parameters in the release 3.1.0. The impact on performance of this growing number of tuning parameters has given rise to a successful line of research on tuning MapReduce systems [17,16,21,18,4,13,15,10,5,24,26,11,12,19].

However, tuning SQL-on-Hadoop queries is more complex than tuning MapReduce jobs, because queries span a workflow of jobs with a single point of configuration (i.e., the tuning setup is set in the query source code or via command line). In practice, developers tune the physical resources to be allocated by the query and the query processing engine propagates such set of physical resources to all jobs in the query plan with potential inefficient performance (see Section 4).

In traditional SQL processing, a query plan indicates the execution flow for the query operators. SQL-on-Hadoop processing is no different, but in the dis-

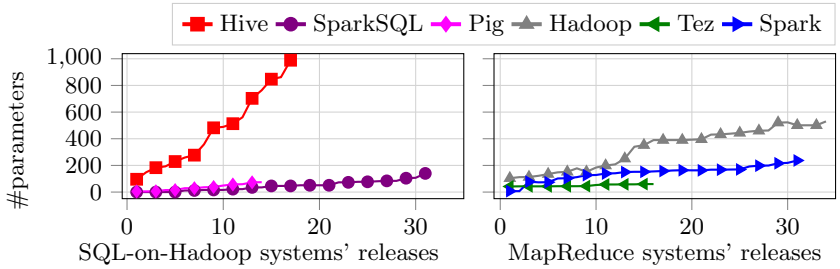


Fig. 1: Number of configuration parameters per release.

tributed environment of MapReduce systems every job of a query plan implements a different set of query operators with different resource consumption needs [6,22]. For instance, while one job requires disk bandwidth due to a *TableScan* operator, another job requires memory throughput due to a *Sort* operator. The lurking problem of tuning SQL-on-Hadoop with a single point of configuration: the propagation of the same set of physical resources to jobs with different resource needs drives the query to inefficient performance.

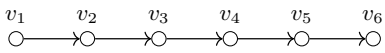
Contribution. In order to avoid such propagation of tuning setup, we propose a novel resource allocation method that assigns a specific tuning setup for each job in the query plan. We formulate the searching for specific tuning setup as a combinatorial optimization problem, highlight its special structure, and show that some special properties preserve the computational efficiency for a particular input of this problem. We also present the shortcoming of tuning SQL-on-Hadoop queries with the current MapReduce tuning advisers and how these queries can achieve better performance when employing our method.

Structure. Section 2 presents the notations and definitions required for presenting the problem. Section 3 presents the problem of tuning SQL-on-Hadoop queries. Section 4 presents the current tuning allocation method and its impact on SQL-on-Hadoop queries. Section 5 presents our method. Section 6 concludes.

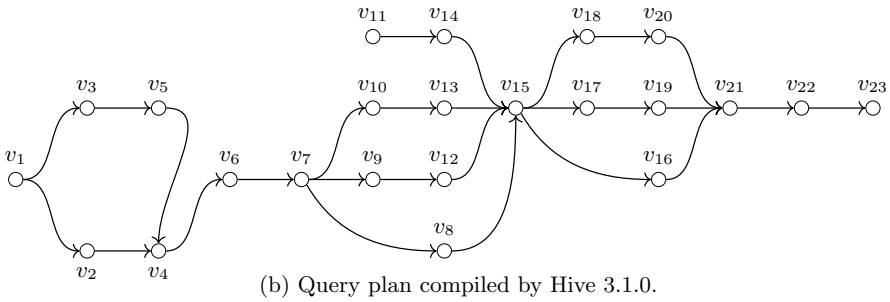
2 Notation and Definitions

Let us define a *query plan* as a directed acyclic graph $G = (V, E)$, where the set of vertices V represents the MapReduce jobs, and the set of edges E denotes the precedence between two jobs. More precisely, a vertex (job) $j \in V$ is a tuple of the form $j = (O_j, T_j, C_j)$ in which O_j is the set of physical *query operators* it executes, T_j is the set of *associated input tables* that are read and processed by j, and C_j is the set of *configurations* used to allocate resources. In this set each $c \in C$ represents a configuration exposed by the SQL-on-Hadoop system that allocates resources to jobs. Each directed edge is an ordered pair of vertices $e = (i, j) \in E$ that connects the jobs i to j, when the execution of i directly precedes j.

Figure 2 illustrates the query plans compiled for TPC-H query 7, in version 0.13.1 and 3.1.0 of Apache Hive to reinforce the growing complexity of SQL-on-



(a) Query plan compiled by Hive 0.13.1.



(b) Query plan compiled by Hive 3.1.0.

Fig. 2: Query plans for TPC-H query 7.

Hadoop queries. Note that the number of MapReduce jobs in each query plan differs considerably across software releases due to the development of different optimization techniques and the addition of new query operators. Although our example sticks to the Apache Hive terminology other SQL-on-Hadoop systems like Spark [3] and Impala [14] also represent the query plans as DAGs.

3 The Tuning Allocation Problem

In SQL-on-Hadoop systems, the task of allocating physical resources through tuning setups mainly comprises two steps: (1) searching for the most efficient tuning setup for a job $j \in V$, and (2) allocating the found tuning setup to the same job j. We define the *most efficient tuning setup* as the configuration that drives the query to decrease response time or minimize resource usage.

Searching for the most efficient tuning setup is a task performed by Tuning Advisers. The common approach is to profile the query execution and, then, to employ heuristics such as Genetic Algorithm [16,17,5], Hill Climbing [15,10], Random Forest [4], Particle Swarm Optimization [13], Exhaustive Search [18], Recursive Random Search [11] and others [21,27]. We define the searching for the most efficient tuning setup as the *Adviser Function*, as follows:

Definition 1 (Adviser Function). *We denote as $f(j)$ the adviser function $f : V \mapsto \mathcal{C}$ which is responsible for always finding the most efficient tuning setup C_j for $j \in V$, where the set $\mathcal{C} = \{C_1, \ldots, C_k\}$ is the collection of possible configurations.*

However, allocating the found tuning setup remains a problem due to the single point of configuration of SQL-on-Hadoop queries. We propose to replace the single point of configuration of queries, exposed to developers, by an automated tuning system able to allocate specific tuning setup per job. Thus, we

define the task of allocating tuning setups C_j for each job $j \in V$ in a query plan G as the problem of assigning the most efficient tuning setup found by the adviser function $f(j)$.

In Definition 2, we formulate the tuning allocation as a combinatorial optimization problem, and use this perspective to develop the reasoning about how to improve the current allocation of configurations to jobs for a query plan. For the problem definition, suppose that we have a computable function $\sigma : \mathcal{C} \times V \rightarrow \mathbb{R}$, where $\sigma(C_i, j)$ determines the computational cost of execute a job j with a tuning setup C_i. We define the problem of assigning the most efficient tuning setup, as follows:

Definition 2 (Tuning Allocation Problem). *Given a directed acyclic graph $G = (V, E)$ representing a query plan, a set $\mathcal{C}$ of possible configurations, and a cost function $\sigma : \mathcal{C} \times V \rightarrow \mathbb{R}$. Find an assignment of tuning setups such that the total cost to process all jobs is minimum, that is, the summation*

$$\sum_{j \in V} \sum_{C_i \in \mathcal{C}} \sigma(C_i, j)$$

is minimum.

Next, we present the current approach employed by SQL-on-Hadoop systems to assign tuning setups to jobs. In Section 5, we present our approach to assign tuning setups to jobs.

4 Uniform Tuning

The methodology employed by the current SQL-on-Hadoop engines allocates the same physical resources to all jobs of a query plan. We name this methodology as the *Uniform Tuning* method, and we treat it as a shortcoming for the query plan. We define the Uniform Tuning method as follows:

Definition 3 (Uniform Tuning). *The Uniform Tuning is the assignment of configurations to jobs such that $C_u = C_v$ for any $u, v \in V$ and $C_u, C_v \in \mathcal{C}$.*

Despite the adviser function $f(j)$ always find the most efficient tuning setup C_j for every job $j \in V$, in the case where only one tuning setup C_j is replicate to all jobs in V (according to Definition 3), C_j may negatively affect the query's performance, as we observe in Figure 3. The uniform tuning problem becomes more severe because SQL-on-Hadoop systems delegate to developers the responsibility to chose one of the available tuning setups C_j to be replicated.

The Fig. 3 illustrates that the Uniform Tuning drives the query processing to inefficient performance in practice through experiments in the Amazon Elastic MapReduce (EMR) cloud environment. We used Starfish [11] to generate the tuning setups. We ran the TPC-H benchmark with Scale Factor of 100GB on Apache Hive (version 0.13.1) and Apache Hadoop (version 0.20.2). We are attached to these versions because they are the versions supported by Starfish.

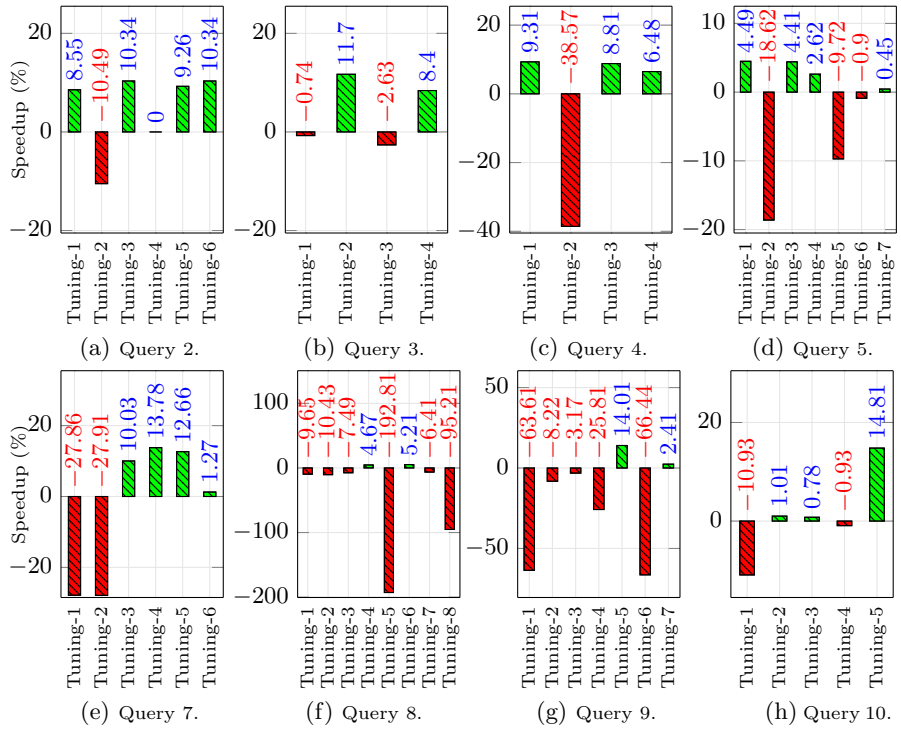


Fig. 3: Speedups compared to default configuration for the uniform tuning. Tuning-{v}, represents a tuning setup generated by Starfish for the job v.

The experiments ran with 6 machines (type c5.2xlarge: 8 of CPU and 16GB of RAM, 100GB of SSD disks). Each runtime is an average of 3 executions.

The x-axis represent the available tuning setups, where tuning-j represents the given C_j generated for a job j. Each C_j was applied uniformly to the query plan, following Definition 3. The y-axis represents the speedups when the execution of the query under the tuning-j is compared to the execution of the default configuration. The default configuration is the configuration bundled with the SQL-on-Hadoop system and is our baseline. According to Definition 3, each tuning setup allocates a different set of resources uniformly to the query, consequently, producing a different outcome. Note that in all queries the performance degrades due to the presence of the uniform allocation of resources, as expected.

We claim that applying a non-uniform allocation method to define a customized tuning setup for each job can outperform the current alternative (uniform tuning), which consists of choosing arbitrarily a tuning setup C_0 and replicate it to all jobs. Therefore, our contribution is a non-uniform allocation method that allows SQL-on-Hadoop systems to apply specific tuning setup per job.

5 Non-Uniform Tuning

For a query plan $G = (V, E)$, every job implements a different set of SQL operators, i.e., for every job $u, v \in V$ we have $O_u \neq O_v$. Consequently, we assume that each job presents different resource consumption needs [6]. For instance, while a job u requires disk bandwidth due to a *TableScan* operator, another job v requires memory throughput due to a *Sort* operator. Next, the rest of this section discuss properties and facts about the particular structure of this problem. The first fact states that the optimum of the problem presented in Definition 2 can be found efficiently when using a non-uniform tuning method. Since this study aims to find a way to do more efficient queries, it is very important to be sure that the allocation of optimal tuning can be made in polynomial time.

The following integer linear programming model describes the optimal solution for the Tuning Allocation Problem. We define the binary decision variable $x_{ij} \in \{0, 1\}$ to indicate whether a tuning setup $C_i \in \mathcal{C}$ is assigned to a job $j \in V$. Let the coefficient $c_{ij} = \sigma(C_i, j)$ be the computational cost for executing a job j with the tuning setup C_i.

$$\text{Minimize} \quad \sum_{j \in V} \sum_{C_i \in \mathcal{C}} c_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{C_i \in \mathcal{C}} x_{ij} = 1 \qquad \forall j \in V \tag{2}$$

$$\sum_{j \in V} x_{ji} = 1 \qquad \forall C_i \in \mathcal{C} \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall C_i \in \mathcal{C}, \forall j \in V \tag{4}$$

The objective function (1) minimizes the computational cost in processing all the jobs for a given query plan. Constraints in (2) ensure that exactly one tuning setup is assigned to a vertex. Reciprocally, the equality in (3) says that only one vertex $j \in V$ can choose a tuning setup $C_i \in \mathcal{C}$. Finally, the constraints in (4) preserve the decision variable's integrality.

Claim 1. The Non-Uniform Tuning method can find the optimal solution for the Tuning Allocation Problem in polynomial time.

The program in (1)-(4) describes precisely the model of an well known combinatorial optimization problem called Assignment Problem. For clarity of exposition, we present a procedure to transform an instance $\langle G = (V, E), \mathcal{C} \rangle$ of the Tuning Allocation Problem into an instance $\langle G' = (V', E') \rangle$ of the original Assignment Problem as follows. Let G' be a undirected graph such that we create a new vertex i associated to each set $C_i \in \mathcal{C}$. Also add a copy of each $j \in V$ and define U as the set of all i added to G' in this way we have $V' = U \cup V$ with U and V disjoint. Now, we add an edge $\{i, j\}$ between every vertex $i \in U$ and $j \in V$. The cost of assigning a tuning setup C_i to job j is c_{ij} and can be interpreted as an weight on the edge $\{i, j\} \in E'$. Notice that all edges

in E' go between U and V. This implies that G' (the input of the Assignment Problem) is a complete bipartite graph.

Even though the Assignment Problem problem is known to be NP-hard, an optimal solution can be obtained efficiently (within execution time bounded by a polynomial function) when the input graph is bipartite [25]. Therefore, the problem can be solved efficiently, because of the *total unimodularity* of the constraint matrix of the Assignment Problem.

An important observation about the Tuning Allocation Problem is that if we drop the constraints (3) of the linear model, the solution remains feasible for the problem. It means that we can attribute a configuration C_i to more than one job (otherwise, the uniform tuning method would not be feasible). Therefore, the natural formulation for this problem is a relaxation for the Assignment Problem. Actually, the resulting integer linear program can be rewritten as follows:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{j \in V} \sum_{C_i \in \mathcal{C}} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{C_i \in \mathcal{C}} x_{ij} = 1 && \forall j \in V \\
& x_{ij} \in \{0, 1\} && \forall C_i \in \mathcal{C}, \forall j \in V.
\end{aligned}$$

Fortunately, the coefficient matrix does not change, and we still have an incidence matrix of a bipartite graph. In this way, the matrix of coefficients of the problem maintains the total unimodularity. Consequently, the Tuning Allocation Problem also can be solved in polynomial time.

Claim 2. For a query plan $G = (V, E)$, in worst case, the non-uniform method is equal to the uniform method, otherwise the non-uniform method performs better.

Let the tuning setup C_0 be the one assigned to all jobs $j \in V$ of the query plan G using the uniform tuning. Now, consider the case with non-uniform tuning in which for the same query plan G, the most efficient tuning for every job j can be obtained by calling interactively the function $f(j)$ (see Definition 1). Directly from definition of the adviser function $f(j)$, we have the following inequality

$$\sigma(f(j), j) \leq \sigma(C_0, j) \tag{5}$$

for every job $j \in V$. The inequality says that, in the best case for the uniform tuning, the arbitrary tuning setup C_0 can be the most efficient for j, but there are no guarantees that it will happen. While attributing a tuning setup C_i chosen by the function $f(j)$, we know that it is always the most suitable tuning setup for j. Consequently, the total cost required to execute all jobs $j \in V$ of the query plan G using the non-uniform method, can be written, as the following summation:

$$\sum_{j \in V} \sigma(f(j), j).$$

So, from the inequality (5) we can compare the total costs of the two methods, as follows:

$$\sum_{j \in V} \sigma(f(j), j) \leq \sum_{j \in V} \sigma(C_0, j),$$

i.e., the computational cost for executing the query plan G using the non-uniform assignment method is at most equals to the cost for the uniform method.

The advantage of using the uniform tuning method is that it is simple and straightforward, i.e., the SQL-on-Hadoop engine just replicates the chosen tuning setup. On the other hand, there are no guarantees that the chosen tuning setup will perform well during the query's execution. In this sense, our proposal of non-uniform tuning personalizes the tuning setup for each job and the optimal assignment of tuning setups is guaranteed. In other words, the Tuning allocation Problem can be solved optimally with the non-uniform tuning method and Claim 1 shows that it can be done in polynomial time. Furthermore, a direct consequence of the non-uniform tuning method is the improvement in the performance of the query plan, as stated in Claim 2.

6 Conclusion

In this paper we presented the shortcoming of MapReduce tuning advisers when applied to SQL-on-Hadoop systems due to the uniform allocation of physical resources and its consequences on query's performance. After, we modeled the Tuning Allocation Problem in order to solve the uniform allocation of physical resources. We presented our *Non-Uniform Tuning* method and proved that it allows assigning optimal tuning setups to SQL-on-Hadoop queries. We also proved that the optimal set of tuning setups required by a query can be found in polynomial time. For future work, an interesting direction consists of combining multiple tuning advisers to optimize one single query, once different tuning advisers focus on different query operators with different resource usage.

Acknowledgement: This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

1. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., Rasin, A.: HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. Proceedings of the VLDB Endowment (2009)
2. Apache Software Foundation: Apache Flink: Stateful Computations over Data Streams. Apache.Org (2015)
3. Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A., Zaharia, M.: Spark SQL: Relational Data Processing in Spark
4. Bei, Z., Yu, Z., Zhang, H., Xiong, W., et al.: RFHOC: A Random-Forest Approach to Auto-Tuning Hadoop's Configuration. TPDS (2016)
5. Bei, Z., Yu, Z., Liu, Q., Xu, C., et al.: MEST: A Model-Driven Efficient Searching Approach for MapReduce Self-Tuning. IEEE Access (2017)

6. Boncz, P.A., Neumann, T., Erling, O.: Tpc-h analyzed: Hidden messages and lessons learned from an influential benchmark. In: TPCTC (2013)
7. Chen, S.: Cheetah: a high performance, custom data warehouse on top of MapReduce. Proceedings of the VLDB Endowment (2010)
8. Costea Adrian Ionescu Bogdan, A.R., Micha Switakowski Cristian Bârc, A., Sompolski Alicja Łuszczak Michał Szafrá nski Giel de Nijs Peter Boncz, J.: VectorH: Taking SQL-on-Hadoop to the next level. In: International Conference on Management of Data - SIGMOD (2016)
9. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI'04 (2004)
10. Ding, X., Liu, Y., Qian, D.: JellyFish: Online Performance Tuning with Adaptive Configuration and Elastic Container in Hadoop Yarn. In: ICPADS (2015)
11. Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L.: Starfish: A Self-tuning System for Big Data Analytics. CIDR (2011)
12. Jiang, D.: The Performance of MapReduce : An In-depth Study. Proceedings of the VLDB Endowment (2010)
13. Khan, M., Huang, Z., Li, M., Taylor, G.A., Khan, M.: Optimizing Hadoop parameter settings with gene expression programming guided PSO. Concurrency and Computation: Practice and Experience (2017)
14. Kornacker, M., Behm, A., Bittorf, V., et al.: Impala: A Modern, Open-Source SQL Engine for Hadoop. CIDR (2015)
15. Li, M., Zeng, L., Meng, S., Tan, J., et al.: MRONLINE: MapReduce online performance tuning. In: HPDC (2014)
16. Liao, G., Datta, K., Willke, T.L., Kalavri, V., et al.: Gunther: Search-based autotuning of MapReduce. Euro-Par (2013)
17. Liu, C., Zeng, D., Yao, H., Hu, C., et al.: MR-COF: A Genetic MapReduce Configuration Optimization Framework. In: Theoretical Computer Science (2015)
18. Liu, J., Ravi, N., Chakradhar, S., Kandemir, M.: Panacea: towards holistic optimization of MapReduce applications. In: CHO (2012)
19. Qin, X., Chen, Y., Chen, J., Li, S., Liu, J., Zhang, H.: The Performance of SQL-on-Hadoop Systems - An Experimental Study. In: IEEE International Congress on Big Data (2017)
20. Sakr, S., Liu, A., Fayoumi, A.G.: The Family of MapReduce and Large-Scale Data Processing Systems. ACM Computing Surveys (2013)
21. Shi, J., Zou, J., Lu, J., Cao, Z., Li, S., Wang, C.: MRTuner: A Toolkit to Enable Holistic Optimization for MapReduce Jobs. PVLDB (2014)
22. Tapdiya, A., Fabbri, D.: A Comparative Analysis of State-of-The-Art SQL-on-Hadoop Systems for Interactive Analytics. In: IEEE International Conference on Big Data (2017)
23. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a Warehousing Solution Over a Map-Reduce Framework. Proceedings of the VLDB Endowment (2009)
24. Van Aken, D., Pavlo, A., Gordon, G.J., Zhang, B.: Automatic Database Management System Tuning Through Large-scale Machine Learning. In: SIGMOD (2017)
25. Wolsey, L.A., Nemhauser, G.L.: Integer and combinatorial optimization. John Wiley & Sons (2014)
26. Yang, H., Luan, Z., Li, W., Qian, D.: MapReduce Workload Modeling with Statistical Approach. Journal of Grid Computing (2012)
27. Zhang, B., Krikava, F., Rouvoy, R., Seinturier, L.: Self-Balancing Job Parallelism and Throughput in Hadoop. In: DAIS (2016)