2015 IEEE 12th International Conference on Autonomic Computing

A Framework For Cost-Effective Scheduling of MapReduce Applications

Nikos Zacheilas, Vana Kalogeraki
Athens University of Economics and Business, Athens, Greece
zacheilas@aueb.gr, vana@aueb.gr

***Abstract*—Real-time, cost-effective execution of "Big Data" applications on MapReduce clusters has been an important goal for many scientists in recent years. The MapReduce paradigm has been widely adopted by major computing companies as a powerful approach for large-scale data analytics. However, running MapReduce workloads in cluster environments has been particularly challenging due to the trade-offs that exist between the need for performance and the corresponding budget cost. Furthermore, the large number of resource configuration parameters exacerbates the problem, as users must manually tune the parameters without knowing their impact on the performance and budget costs. In this paper, we describe our approach to cost-effective scheduling of MapReduce applications. We present an overview of our framework that enables appropriate configuration of parameters to detect cost-efficient resource allocations. Our early experimental results illustrate the working and benefit of our approach.**

I. Motivation

In recent years we observe an increased demand for processing large amounts of data. Systems such as Google's MapReduce [1] and Spark [2] have quickly become de facto big data processing frameworks as they provide powerful and cost-effective approaches. In particular, Hadoop [3], which is MapReduce's most commonly used open-source implementation, can scale out to thousands of nodes and process petabyte data. Due to its high scalability and performance, Hadoop has gained much popularity and is used by many companies, including Amazon [4] and Facebook [5] for running their applications on clusters. For example, Facebook collects 15 TeraBytes of data each day into its PetaByte-scale data warehouse, applying ad hoc analysis and business-intelligence applications utilizing Hadoop [6].

As MapReduce clusters are increasingly shared among multiple users who concurrently submit their MapReduce jobs, cloud providers author users control over the amount of resources to be used by charging them based on the processing and storage resources they bind [7]. For example, in Amazon, users are charged based on the hours they bind their allocated Virtual Machines and the amount of I/O operations performed by their applications [4]. Thus, a challenging problem for the user is to efficiently decide how many resources to allocate for her jobs in order to satisfy her requirements (e.g. minimize jobs' makespan) without overspending. The problem is exacerbated by the fact that the MapReduce job performance can be affected by a wide range of configuration parameters [8]. Currently the burden of configuring these parameters falls on the user who submits the jobs.

Work has been done with respect to allocating resources to a user's jobs, trying to satisfy each job's execution time requirements [9]. However, this scheme focuses on single jobs and is not optimized when multiple jobs execute concurrently. Our previous work [10] proposed a laxity-based scheduler for specifying the execution order of jobs but did not consider the user's spending budget. There has also been some recent proposals regarding the automatic parameter tuning in Hadoop, specifically Starfish [8] and MRTuner [11]. However, these approaches do not examine the implications when multiple jobs execute concurrently in the cluster (i.e. how map/reduce slots should be allocated between these jobs to satisfy both budget and performance constraints). In this paper, we describe our approach to cost-effective scheduling of MapReduce applications. We present an overview of our framework that enables appropriate configuration of parameters to detect cost-efficient resource allocations. Our early experimental results illustrate the working and benefit of our approach.

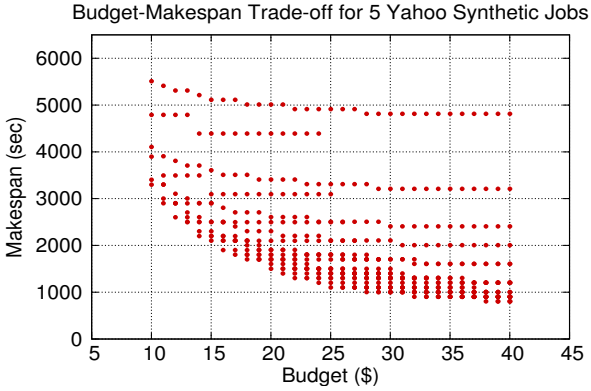

Fig. 1. Budget-Execution Time Trade-off Problem

II. Framework Description

A MapReduce job or application is modelled as a sequence of two computational phases, the map and the reduce phase. Each phase consists of multiple tasks that execute in parallel. Tasks are modelled as follows: $map(k_1, v_1) \Rightarrow [k_2, v_2]$ and $reduce(k_2, [v_2]) \Rightarrow [k_3, v_3]$. . Map tasks take as input (k_1, v_1) pairs and return a list of intermediate *(key,value)* pairs of possibly different types, k_2 and v_2. The values associated with the same key k_2 are grouped together into a list and are processed by the appropriate reduce task. The execution time of a job is equal to the sum of the execution times of the map and the reduce phases. The maximum number of map and reduce tasks that can run concurrently in the cluster depends on the available resources (i.e. map/reduce slots). Usually users submit multiple jobs concurrently (i.e. MapReduce workload) for execution in the cluster.

978-1-4673-6971-8/15 $31.00 © 2015 IEEE
DOI 10.1109/ICAC.2015.38

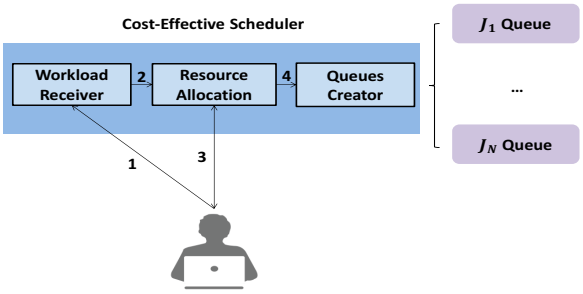

Fig. 2. Framework's Overview

Based on the amount of resources (i.e. map/reduce slots) that the jobs reserve, the user is charged with the corresponding monetary cost. We define the cost similarly to [7] as the number of reserved slots multiplied by the per slot monetary cost (i.e. a metric defined by the cloud provider). Furthermore, users want to minimize the end-to-end execution time (i.e. makespan) of their workload. Therefore we have a a multi-objective optimization problem that needs to be solved, as we have to minimize the workload's makespan and at the same time minimize their spending budget. In Figure 1 we illustrate the problem with a synthetic MapReduce workload. Our workload comprised of 5 synthetic Yahoo jobs with their characteristics defined in [12]. We assumed that the user's budget is between 10$ and 40$. As you can see in Figure 1 based on the different combinations of map/reduce slots' allocations we end up with different budget and workload's makespan. Assigning more slots to the jobs decreases the observed makespan but at the same time leads to an increase in the user's spending budget.

We illustrate the high level overview of our framework in Figure 2. The user provides the range of budget she is willing to spend for the execution of a set of jobs, and then the system suggests resource allocation plans (i.e. map/reduce slots allocations) that are optimal in regards to the budget/makespan trade-off. Furthermore, our framework examines the impact of the reduce task configuration parameter and adjusts it to further minimize the job's execution time. We illustrate the benefits of tuning this parameter in Figure 3 where we display the impact of the reduce tasks in the observed execution time of a Sorting job running in our 7-VM cluster which consists of 14 map and 14 reduce slots. The job sorts 5 GB of randomly generated data. One can observe that by increasing the reduce tasks beyond a certain value provides small benefits. In this case our framework would detect that the best solution is when we use the same number of reduce tasks as the available reduce slots and would apply this setting to the user's job.

Our system applies a slots' allocation by initiating a separate scheduling queue for each job. Each queue will have the configured number of map/reduce slots for the corresponding job (i.e. Queues Creator in Figure 2). This way we limit the slots used by each job only to its allocated value. We use Hadoop's FAIR [3] scheduler for the actual execution of the jobs in the cluster. The scheduler assigns each job to its queue, so all jobs can execute concurrently, utilizing their allocated map/reduce slots. We do not apply the default Hadoop FIFO scheduler as it does not offer a way to limit the resources used by each job, but rather assumes that all the resources are

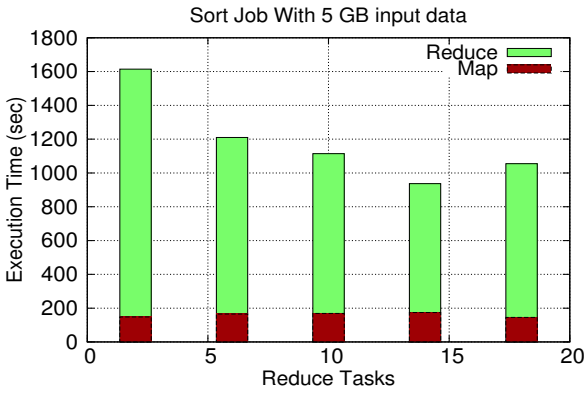

Fig. 3. Reduce Task Adjustment

available, so a job may occupy all the slots in the cluster. Our approach can be easily extended for other Hadoop queue-based schedulers such as the Capacity Scheduler [3].

III. Conclusions

In this work, we present a novel framework that balances the trade-off between budget and performance for concurrently running MapReduce applications. Our initial experimental results indicate the benefits of our proposal by adjusting basic configuration parameters.

IV. Acknowledgments

This research has been financed by the European Union through the FP7 INSIGHT project and the ERC IDEAS NGHCS project.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] Spark, http://spark.apache.org.

[3] Hadoop, http://hadoop.apache.org.

[4] Amazon EC2, http://aws.amazon.com/ec2/.

[5] Facebook, http://www.facebook.com.

[6] 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," *PVLDB, Pages 1626-1629*, 2009.

[7] T. Sandholm and K. Lai, "Dynamic Proportional Share Scheduling in Hadoop," *JSSPP'10, Pages 110-131, Atlanta GA, USA, April*, 2010.

[8] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of mapreduce programs," *VLDB Endowment*, vol. 4, no. 11, pp. 1111–1122, 2011.

[9] A. Verma, L. Cherkasova, V. S. Kumar, and R. H. Campbell, "Deadline-based Workload Management for MapReduce Environments: Pieces of the Performance Puzzle," *NOMS, Page(s): 900 - 905, 16-20 April*, 2012.

[10] N. Zacheilas and V. Kalogeraki, "Real-time scheduling of skewed mapreduce jobs in heterogeneous environments," in *ICAC*, Philadelphia, PA, Jun. 2014, pp. 189–200.

[11] J. Shi, J. Zou, J. Lu, Z. Cao, S. Li, and C. Wang, "MRTuner: A Toolkit to Enable Holistic Optimization for MapReduce Jobs," *Proceeding of the VLDB Endowment 7(13): 1319-1330*, 2014.

[12] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *CCGrid*, Melbourne, May, 2010, pp. 94–103.