Proceedings of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

A Control Approach for Performance of Big Data Systems

M. Berekmeri [*,**,****,***] D. Serrano [**,****,***]
S. Bouchenak [**,****,***] N. Marchand [*,****,***] B. Robu [*,****,***]

[*] *GIPSA-lab, BP46 38402 Grenoble, France*
email: {mihaly.berekmeri, nicolas.marchand, bogdan.robu}@gipsa-lab.fr
[**] *Distributed Computer Systems Group, LIG, BP53 38402 Grenoble, France*
email: {damian.serrano, sara.bouchenak}@imag.fr
[***] *CNRS, France*
[****] *Univ. Grenoble Alpes, F-38402 Grenoble, France*

Abstract: We are at the dawn of a huge data explosion therefore companies have fast growing amounts of data to process. For this purpose Google developed MapReduce, a parallel programming paradigm which is slowly becoming the *de facto* tool for Big Data analytics. Although to some extent its use is already wide-spread in the industry, ensuring performance constraints for such a complex system poses great challenges and its management requires a high level of expertise. This paper answers these challenges by providing the first autonomous controller that ensures service time constraints of a concurrent MapReduce workload. We develop the first dynamic model of a MapReduce cluster. Furthermore, PI feedback control is developed and implemented to ensure service time constraints. A feedforward controller is added to improve control response in the presence of disturbances, namely changes in the number of clients. The approach is validated online on a real 40 node MapReduce cluster, running a data intensive Business Intelligence workload. Our experiments demonstrate that the designed control is successful in assuring service time constraints.

Keywords: disturbance rejection, linear control systems, control for computers, cloud computing, Big Data

1. BACKGROUND AND CHALLENGES

As we enter in the era of Big Data (Big Data refers to a collection of data sets so large and complex that it becomes difficult to process using traditional database management tools), the steep surge in the amount data produced brings new challenges in data analysis and storage. Recently, there is a growing interest in key application areas, such as real-time data mining, that reveals a need for large scale data processing under performance constraints. These applications may range from real-time personalization of internet services, decision support for rapid financial analysis to traffic controllers. The steep increase in the amount of unstructured data available therefore calls for a shift in perspective from the traditional database approach to an efficient distributed computing platform designed for handling petabytes of information. This imposes the adaptation of internet service providers to implementations on distributed computing platforms and one way to achieve this is to adopt the popular programming model called **MapReduce**. Its success lies in its usage simplicity, its scalability and fault-tolerance. MapReduce is backed and intensively used by the largest industry leaders such as Google, Yahoo, Facebook and Amazon. As an illustration, Google executes more than 100.000 MapReduce jobs every day, Yahoo has more the 40.000 computers running MapReduce jobs and Facebook uses it to analyse more then 15 petabytes of data. The MapReduce programming paradigm was initially developed by Google in 2008 as a general parallel computing algorithm that aims to automatically handle data partitioning, consistency and replication, as well as task distribution, scheduling, load balancing and fault tolerance (see Dean and Ghemawat (2008) for further details).

In the same time, there is a growing interest of computer science researchers in control theory to automatically handle configurations of complex computing systems. Recent publications in the field of continuous time control of computer systems show the emergence of this new field for automatic control. For instance, continuous time control was used to control database servers (Malrait et al., 2009) using Lyapunov theory, web service systems (Poussot-Vassal et al., 2010) or HTTP servers (Hellerstein et al., 2004) using a "blackbox" approach. It must be underlined that this field is also emerging in the field of discrete event systems, see Rutten et al. (2013) for a survey.

The aim of this paper is to propose a control based approach to tune MapReduce. MapReduce is a way to implement internet programs and to run them in a parallel way on many computers in the cloud (called nodes). Although MapReduce hides most of the complexity of parallelism from users [1], deploying an efficient MapReduce implementation still requires a high level of expertise. It is for instance the case when tuning MapReduce's

[⋆] This work has been supported by the LabEx PERSYVAL-Lab 5ANR-11-LABX-0025.

[1] By MapReduce users we mean companies wishing to use MapReduce for their own applications, typically internet services providers.

configuration as underlined in (White, 2012; Herodotou and Babu, 2011) or to assure performance objectives as noted in (Xie et al., 2012). By performance objective, we usually mean the service time, that is the time needed for the program running on the cloud to serve a client request. For a user to run a MapReduce job at least three things need to be supplied to the framework: the input data to be treated, a Map function, and a Reduce function. From the control theory point of view, the Map and Reduce functions can be only treated as black box models since they are entirely application-specific, and we assume no *a priori* knowledge of their behavior. Without some profiling, no assumptions can be made regarding their runtime, their resource usage or the amount of output data they produce. On top of this, many factors (independent of the input data and of the Map and Reduce functions) influence the performance of MapReduce jobs: CPU, input/output and network skews (Tian et al., 2009), hardware and software failures (Sangroya et al., 2012), Hadoop's (Hadoop is the most used open source implementation of MapReduce) node homogeneity assumption not holding up (Zaharia et al., 2008; Ren et al., 2012), and bursty workloads (Chen et al., 2012). All these factors influence the MapReduce systems as perturbations. Concerning the performance modelling of MapReduce jobs, the state of the art methods use mostly job level profiling. Some authors use statistical models made of several performance invariants such as the average, maximum and minimum runtimes of the different MapReduce cycles (Verma et al., 2011). While others employ a static linear model that captures the relationship between job runtime, input data size and the number of map, reduce slots allocated for the job (Tian and Chen, 2011). In both cases the model parameters are found by running the job on a smaller set of the input data and using linear regression methods to determine the scaling factors for different configurations. A detailed analytical performance model has also been developed for off-line resource optimization, see Lin et al. (2012). Principle Component Analysis has also been employed to find the MapReduce/Hadoop components that most influence the performance of MapReduce jobs (Yang et al., 2012).
It is important to note that **all the presented models predict the steady state response of MapReduce jobs and do not capture system dynamics**. They also assume that **a single job is running at one time in a cluster**, which is far from being realistic. The performance model that we propose addresses both of these issues: it deals with a concurrent workload of multiple jobs and captures the systems dynamic behaviour.

Furthermore, while MapReduce resource provisioning for ensuring Service Level Agreement (SLA) [2] objectives is relatively a fresh area of research, there are some notable endeavours. Some approaches formulate the problem of finding the optimal resource configuration, for deadline assurance for example, as an off-line optimization problem, see Tian and Chen (2011) and Zhang et al. (2012). However, we think that off-line solutions are not robust enough in real life scenarios. Another solution is given by ARIA, a scheduler capable of enforcing on-line SLO deadlines. It is build upon a model based on the job completion times of past runtimes. In the initial stage an off-line optimal amount of resources are determined and then an on-line correction mechanism for robustness is deployed. The control input they choose is the number of slots given to a respective job. This is a serious drawback since the control works only if the cluster is sufficiently over-provisioned and there are still free slots to allocate to the job. Another approach is SteamEngine developed by Cardosa et al. (2011) which tries to avoid the previous drawback and dynamically add and remove nodes to an existing cluster.
However, **in all the previous cases, it is assumed that every job is running on an isolated virtual cluster and therefore they don't deal with concurrent job executions**.
Taking all these challenges into consideration our contributions are two fold: we developed the **first dynamic model for MapReduce** systems and we built and implemented the **first on-line control framework** capable of assuring service time constraints for a concurrent MapReduce workload.

2. OVERVIEW OF BIG DATA

2.1 MapReduce Systems

MapReduce is a programming paradigm developed for parallel, distributed computations over large amounts of data. The initial implementation of MapReduce is based on a master-slave architecture. The master contains a central controller which is in charge of task scheduling, monitoring and resource management. The slave nodes take care of starting and monitoring local mapper and reducer processes.
One of its greatest advantages is that, when developing a MapReduce application, the developer has to implement only two functions: the Map function and the Reduce function. Therefore, the programmers focus can be on the task at hand and not on the messy overhead associated with most of the other parallel processing algorithms, such as is the case with the Message Parsing Interface protocol for example.
After these two functions have been defined we supply to the framework our input data. The data is then converted into a set of (key,value) pairs. The Map functions take the input sets of (key,value) pairs and output an intermediate set of (key,value) pairs. The MapReduce framework then automatically groups and sorts all the values associated with the same keys and forwards the result to the Reduce functions. The Reduce functions process the forwarded values and give as output a reduced set of values which represent the answer to the job request.
The most used open source implementation of the MapReduce programming model is Hadoop. It is composed of the Hadoop kernel, the Hadoop Distributed Filesystem (HDFS) and the MapReduce engine. Hadoop's HDFS and MapReduce components are originally derived from Google's MapReduce and Google's File System initial papers (Dean and Ghemawat, 2008). HDFS provides the reliable distributed storage for our data and the MapReduce engine gives the framework with which we can efficiently analyse this data, see White (2012).

2.2 Experimental MapReduce Endvironment

The MapReduce Benchmark Suite (MRBS) developed by Sangroya et al. (2012) is a performance and dependability benchmark suite for MapReduce systems. MRBS can emulate several types of workloads and inject different fault types into a MapReduce system. The workloads emulated by MRBS were selected to represent a range of loads, from the compute-intensive to the data-intensive (e.g. business intelligence - BI) workload. One of the strong suites of MRBS is to emulate client requests. One request may consist of one or more MapReduce

[2] SLA is as a part of a service contract where services are formally defined.