Feedback Autonomic Provisioning for Guaranteeing Performance in MapReduce Systems

Mihaly Berekmeri, Damián Serrano, Sara Bouchenak, Nicolas Marchand, Bogdan Robu

Feedback Autonomic Provisioning for Guaranteeing Performance in MapReduce Systems

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

Abstract—Companies have a fast growing amounts of data to process and store, a data explosion is happening next to us. Currently one of the most common approaches to treat these vast data quantities are based on the MapReduce parallel programming paradigm. While its use is widespread in the industry, ensuring performance constraints, while at the same time minimizing costs, still provides considerable challenges. We propose a coarse grained control theoretical approach, based on techniques that have already proved their usefulness in the control community. We introduce the first algorithm to create dynamic models for Big Data MapReduce systems, running a concurrent workload. Furthermore, we identify two important control use cases: relaxed performance - minimal resource and strict performance. For the first case we develop two feedback control mechanism. A classical feedback controller and an even-based feedback, that minimises the number of cluster reconfigurations as well. Moreover, to address strict performance requirements a feedforward predictive controller that efficiently suppresses the effects of large workload size variations is developed. All the controllers are validated online in a benchmark running in a real 60 node MapReduce cluster, using a data intensive Business Intelligence workload. Our experiments demonstrate the success of the control strategies employed in assuring service time constraints.

Index Terms—Control for computing systems, event based control, cloud computing, feedforward control, Big Data

1 Introduction

1.1 Background and challenges

We are at the dawn of a data and computing revolution. The amount of raw data produced by everything from our mobile phones, tablets, computers to our smart watches is increasing exponentially. As a result companies face novel and growing challenges in data storage and analysis. The sheer amount of data available is asking for a shift of perspective from the traditional database approaches to platforms capable of handling petabytes of unstructured information available for tasks such as personalized advertising, advanced data mining or classification.

One of the most popular of such platforms is the MapReduce framework, which is one of the currently most utilised programming paradigm in use for parallel, distributed computations over large amounts of data. MapReduce is backed by the largest BigData industry leaders. For example, Google has more than 100 thousand MapReduce jobs executed daily [1] , Yahoo has more than 40 thousand computers running MapReduce jobs, Linkedin evaluates more than 120 billion relationships per day using MapReduce [2] while, Facebook's largest MapReduce cluster contains more than a 100 petabytes of data.

Nevertheless, while current commercial MapReduce services such as Amazon EMR [3] and Microsoft HDInsight [4] offer solutions for quick and cost-effective Big Data processing they don't provide any guarantees in terms of application performance. Furthermore, while some elasticity mechanism are given, they are not completely autonomous and several important scaling decisions, such as selecting the different scaling thresholds, are left up to the service user.

However, before such autonomous control solutions can be synthesised, performance models need to be built that can capture the dynamic behaviour of a MapReduce system. These models can constitute the basis upon which an automatic controller can decide when and how to optimally intervene in the system in order to keep the desired Quality of Service (QoS). This QoS is formalised in the cloud in the form of a Service Level Agreement (SLA), which is a contract negotiated between the clients and their service provider. An SLA can consist of multiple Service Level Objectives (SLOs), for example the maximum service time to be guaranteed by the provider.

SLAs are a relatively a fresh area of research in cloud systems, for more details see [5], [6], [7]. Although current Big Data MapReduce solutions do not provide guarantees in terms of performance and/or dependability, we believe that more and more customers will be interested on having such guarantees and that those service providers that can provide them, will gain

- *M. Berekmeri, B. Robu, N. Marchand are with GIPSA-lab, BP46 38402 Grenoble, France. They are also with CNRS, France and Univ. Grenoble Alpes, F-38402 Grenoble, France E-mail: mihaly.berekmeri@gipsa-lab.fr*
- *D. Serrano is with IRISA Research Group, University of Rennes 1, France. He is also with CNRS.E-mail: damian.serrano@irisa.fr*
- *S. Bouchenak is with LIRIS Research Group, LIRIS UMR 5205, INSA de Lyon, France. She is also with CNRS, France and Univ. INSA-Lyon, 69621 Villeurbanne, France.E-mail: sara.bouchenak@insa-lyon.fr*

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

a competitive advantage, see European Projects MyCloud [8], HARNESS [9].

But why is the performance modelling and control of a MapReduce service such a challenge? If one desires 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. However, the Map and Reduce functions can only be treated as black box models since they are entirely application-specific, and we have no *a priori* knowledge of their behaviour. Without some profiling, no assumptions can be made regarding their runtime, resource usage or the amount of output data produced. On top of this, many other independent factors have been identified that influence the performance of MapReduce jobs: CPU, input/output and network skews [10], hardware and software failures [11], Hadoop's (Hadoop is the most used open source implementation of MapReduce) node homogeneity assumption not holding up [12], [13], and bursty workloads [14] and [15].

Moreover, when it comes to the cloud, resource provisioning for deadline management is further made difficult because of the shared hardware resource architecture, where interference and concurrency issues may arise frequently. Furthermore, as cloud providers desire to maximise the resource utilisation of their clusters, they have mechanisms for the dynamic reallocation of unused resources which further adds to the variability of system performance. So even with the same workload and resource amount, an application performance may vary depending on how noisy neighbouring applications are. In the meantime, for most businesses of course, missing deadlines results in financial losses. In some cases these costs can go up to 100.000$ per minute, as is the case of an on-line brokerage industry [16].

As a result lots of research is being done in the HPC, Grid, Database communities on improving the performance, dependability of complex computing systems such as MapReduce. Extensive research has been conducted already to improve upon MapReduce [17], [18], [19] by changing the behaviour and algorithms of the MapReduce framework itself. A key point to make here is that, although these solutions improve upon how MapReduce works, no performance guarantees are provided.

In addition, our work differentiates itself from these in several other aspects. First of all, we present a novel method that enables the simplified automatic modelling of complex distributed systems. We chose MapReduce systems as our test case because it is a highly dynamic system in both data quantity, richness and in terms of its processing needs and it is one of the most popular current architectures for distributed data processing.

Furthermore, one can notice that, due to the unpredictability of new deployment environments, such as the cloud, traditional adaptation approaches become increasingly difficult to use. Therefore, more and more attention is given to approaches used in different fields for controlling complex systems. The most prominent of these are the feedback control solutions coming from the field of control theory [20], [21], [22], [23], [24], which has been providing control solutions for physical systems for several decades now. The advantages of control theory are that it can provide a solid mathematical basis for synthesizing feedback control loops, for handling safely complexity and for having theoretically guaranteed results.

Our work is in line with these latter approaches as we develop on-line feedback, feedforward control techniques that don't require complex tuning and that, contrary to existing heuristic approaches, can give theoretically guaranteed performance. Moreover, our approach is non-intrusive, it does not modify the framework. Which, together with the generality of the developed techniques presented in this paper, allows for the applicability of our approach to a wide variety of cloud systems.

1.2 Scientific contributions

Taking all this into consideration, in this paper, we propose a new approach to the performance modelling and control of Big Data MapReduce cloud services. Our contributions in this paper are the following:

- We provide the first algorithm for building dynamic models for Big Data MapReduce systems
- We develop and implement multiple novel controllers able to assure service time constraints for a concurrent MapReduce workload in two different industrial scenarios: in the first case the assurance of relaxed performance constraints with minimal resource usage is desired, while in the second one strict performance constraints are given, allowing for a large resource consumption for short periods of time.

1.3 Paper roadmap

The remainder of the paper is organized as follows. Section 2 gives a brief overview of Big Data MapReduce. Section 3 presents our contribution in constructing an algorithm for building dynamic models for Big Data MapReduce systems and its experimental validations. In Section 4 the proposed control architecture is introduced. Section 5 contains the description of two separate relaxed performance - minimal resource control laws and their experimental validation. Section 6 contains the description of the strict performance control law and its experimental validation. The related work is described in Section 7. Finally, Section 8 draws the conclusions and presents our ideas for future work.

2 Background

2.1 MapReduce Systems

The main objectives of Big Data Clouds are to capture, store, analyse and manipulate large and complex

amounts of data. MapReduce is one of the currently most used programming paradigms 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 [25].

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 originally derived from Google's MapReduce and Google's File System initial papers [1]. HDFS provides reliable distributed storage for our data and the MapReduce engine gives the framework with which we can efficiently analyse this data, see [25].

When it comes to MapReduce performance modelling, the state of the art methods use mostly job level profiling [26], [27], [28]. Moreover it is important to note that *current models predict only the steady state performance* of MapReduce jobs and do not capture system dynamics. They also assume that a *single job is running at one time in a cluster*, neglecting any workload fluctuations and interferences. Meanwhile, a recent survey among BigData solution providers (Teradata) and consumers (Ebay, Facebook) reveals the unanimous response "Real Big Data clusters are never run in single-user mode - they never run just one job at a time" [29]. Still, in the case of cloud deployments, the resource usage being on demand, resources are mostly provisioned per MapReduce job execution at the moment. We believe that this is changing partly because of financial reasons. On one hand, most cloud providers provide incentives for long term reservation of nodes with pricing up to 60% less than on demand rates, which is very advantageous for batch jobs. On the other hand, in our experiments we have observed that just by running multiple jobs on the same cluster, instead of separate ones, one can save up to 66% of the deployment costs. Therefore, to address these issues a performance model is proposed that captures the dynamic behaviour of a concurrent workload of multiple jobs.

2.2 Experimental MapReduce Endvironment

All the experiments in this paper were conducted on-line in Grid5000, on a single cluster of 60 nodes. The 60 nodes infrastructure was chosen for practical reasons, as we don't yet have access to a larger cluster size. However, all algorithms presented in this work scale well, and can be applied to any cluster size, with only the re-identification of the equation parameters as described in Section 3.5.

Grid5000 is a French nation-wide cluster infrastructure made up of 5000 CPUs, developed to aid parallel computing research. It provides a scientific tool for running large scale distributed experiments, see [30]. Each node from the cluster used for our experiments has a quad-core Intel CPU of 2.53GHz, an internal RAM memory of 15GB, 298GB disk space and the connection between the nodes is assured with an Infiniband 20G network.

For our experiments we use the open source MapReduce implementation framework Apache Hadoop v1.1.2 [31] and the high level MRBS benchmarking suite. A data intensive BI workload is selected as our workload. The BI benchmark consists of a decision support system for a wholesale supplier. Each client interaction emulates a typical business oriented query run over a large amount of data (*10GB* in our case). To generate the client interactions Apache Hive is deployed on top of Hadoop, which converts SQL like queries to a series of MapReduce jobs. All the nodes is the cluster were on the same switch to minimize network skews.

The MapReduce Benchmark Suite [11] (MRBS) 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 are designed to cover five application domains: recommendation systems, business intelligence (BI), bioinformatics, text processing and data mining. These workloads were selected to represent a range of loads, from the compute-intensive (e.g. recommendation systems) to the data-intensive (e.g. business intelligence - BI) workload. One of strengths of MRBS is to emulate client interactions (CIs), which may consist of one or more MapReduce jobs. These jobs are the examples of what may be a typical client interaction within a real deployment of a MapReduce system.

A simplified version of our experimental setup is sketched in Figure 1. We measure from the cluster the service time [1] and the number of clients and we use the number of nodes in the cluster to ensure the service time deadlines, regardless the changes in the number of the clients.

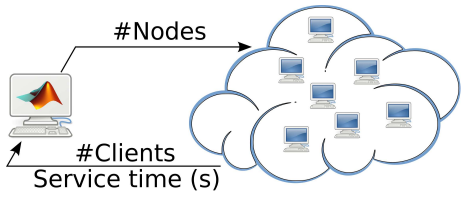


Fig. 1. Intuitive view of the experimental setup

The control strategy is implemented in Matlab and all the measurements are made online, in real time. Although Matlab already provides an array of general

1. By service time we mean the time it takes for a client interaction to run. In computer science this is also known as response time. We did not use this term since in the control theory community "response time" means something completely different.

controllers, all the control algorithms presented here were implemented from scratch, and were specifically designed for our set-up. We use Matlab as it is the standard tool in control theory, but all of the algorithms could have been easily implemented in any other programming language, like C++, Java for example. Where we exploited Matlab's powerful tools were the initial simulations and during the model identification phase. All our actuators and sensors are implemented in Linux Bash scripts. For a more detailed version of our experimental setup one can check Figure 2.

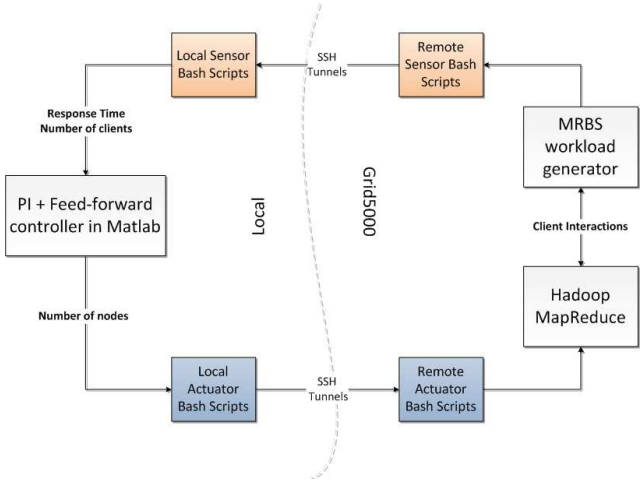


Fig. 2. Detailed view of the experimental setup

The procedure of an experiment run is the following. At the beginning of the experiment two communication channels are created. These are used to send the control commands, that can remove or add nodes to the Hadoop cluster, and to monitor the runtime performance metrics (e.g. service time) through our custom monitoring tools, which periodically process Hadoop's logs. Since the monitoring information for different jobs can be found on the master node, slave level monitoring is not required.

3 MapReduce performance model

3.1 Choosing the model inputs/outputs

The choice of *control inputs* out of Hadoop's many parameters (more than 170) is not straightforward. As we set out for our model to be implementation agnostic, we take into consideration only those parameters that have a high influence regardless of the MapReduce version used. Two such factors that have been identified having among the highest influence are the number of Mappers and the number of Reducers available to the system, see [32]. As these parameters are fixed per node level we chose the number of nodes to be our control input since it effects both.

Our *control output* is the service time, defined as the average time (y) needed to process requests in a certain time window. Low client service time is a desirable as it reflects a reactive system.

$$y[s] = avg(y_1, y_2, ..., y_N) \quad (1)$$

The average y is calculated at every 30 seconds, using a sliding window of $T = 15$ minutes length.

3.2 Analysing system behaviour

Let us now present the behaviour of the system in case of variations in the number of nodes (Fig. 3) and respectively clients (Fig. 4). Each figure presents the results after a warm-up phase of 20 minutes in which the system reaches nominal operation.

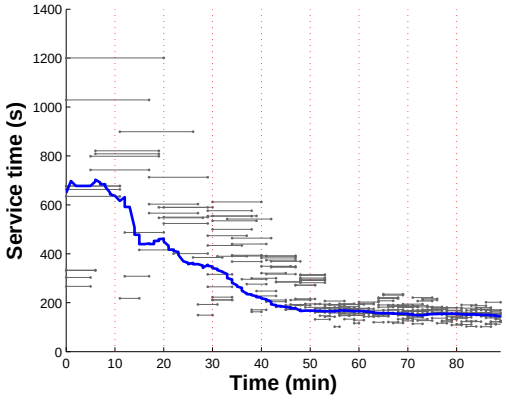
The first experiment presents the results when the number of nodes increases from 4 to 36. The number of concurrent clients is fixed to 10 during the whole experiment and 4 more nodes are added every 10 minutes, see Fig. 3.(b). The multiple horizontal lines in Fig. 3.(a) constitute the job runtimes. The jobs are grouped together vertically based on their runtime. It can be seen that the overall behaviour of MapReduce is non-linear since a proportional increase in the number of nodes is not proportional to the reduction of response time (see Figure 3 below).

In the second experiment we analyse the behaviour of the MapReduce system for a fixed number of nodes (20 in our case) and for an increasing number of clients. The number of clients is increased by 5 every 10 minutes from 5 clients to 40, see Fig. 4.(b). We can see in Figure 4.(a) that the behaviour is also non-linear. It is also interesting to notice that, although the throughput gets saturated in this case as well, it is not for the same reasons as in the previous case, when we had more resources then required by the current workload. In contrast, here the throughput is saturated when we have a lack of resources.

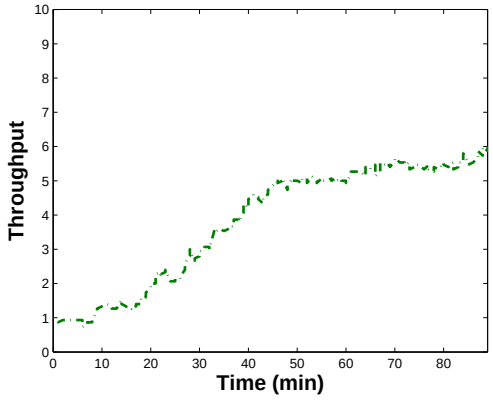
Although the overall system behaviour is non-linear, in nominal operation (when the system is not highly under- or overloaded), the system behaviour can be approximated with a linear model, which is found by linearising around a chosen operating point defined by a baseline number of nodes and clients. Linearisation is generally preferred when possible, because a linear model is less complex to build and adapt on-line. Furthermore, one can take advantage of the numerous control synthesis methods that have been already developed for linear systems.

Our proposed methodology for choosing the linearisation point is the following. After the client decides on the number of nodes to have serving request (usually based on financial constraints since the client rents the nodes from the service provider) our algorithm gradually increases the number of clients, until the throughput of the cluster is maximized[2] (it is important to maximise throughput for both environmental and financial

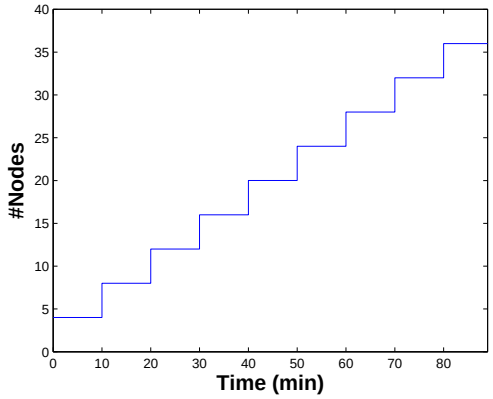
2. Throughput is measured as the number of client connection demands per second and it has *an inverse relation with service time*



(a) Service time

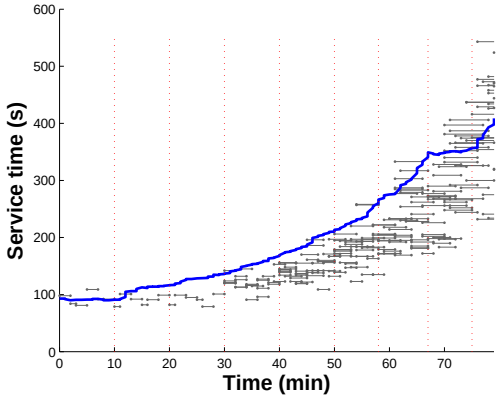


(b) Throughput

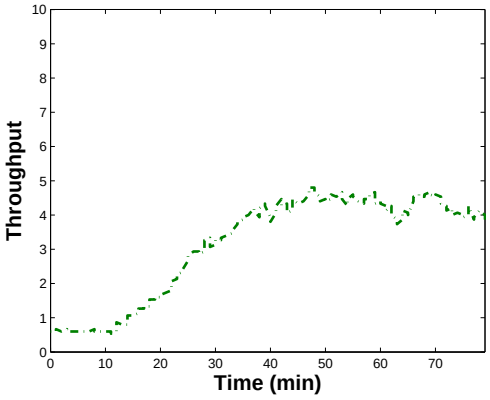


(c) Nodes

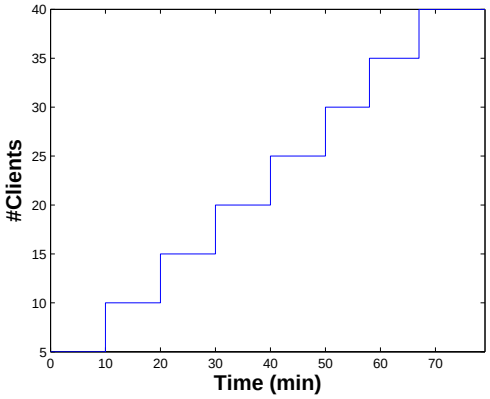
Fig. 3. Effects of nodes variation, #Clients=10



(a) Service time



(b) Throughput



(c) Clients

Fig. 4. Effects of workload variation, #Nodes=20

reasons). This point of full utilization will be the set-point for linearisation. More details on this issue are given below in Section 3.6 where a general algorithm for building dynamic performance BigData models is given.

For this specific case, we find this point from the previous experiments shown in Figure 3.(b) and Figure 4.(b). As it can be seen, the point where the throughput starts to saturate, is at 20 nodes and 10 clients, which will be our set-point for linearisation.

3.3 Capturing system dynamics

One of the important challenges in current MapReduce deployments is assuring certain service time thresholds for jobs. Therefore, our control objective is selected as keeping the average service time below a given threshold for jobs that finished in a the last time window. This time window is introduced only to assign a measurable dynamics to the system. The definition of the time window length is not straightforward. The bigger the window is,

the more we loose system dynamics, while the smaller it is, the bigger the noise will be in the measurements.

In our case the window size is tuned off-line. To choose the windows size we start with a small value and increase it gradually until we reach the desired signal variance and the curves smoothen out. Let us now remind that below this size the output measurements may be influenced by the noise that arises from the natural variance of the jobs. From a control perspective, if the window is bigger than this size then the controller reacts slower and if it is smaller it will react to noise.

3.4 Proposed model structure

The high complexity of a MapReduce system and the continuous changes in its behaviour (because of software upgrades and improvements) prompted us to avoid the use of white-box modelling and to opt for a technique which is agnostic to these issues. This leads us to a grey-box or black-box modelling technique. The line between these two techniques is not well defined, but we consider our model a grey-box model since the structure of the model was defined based on our observations of linearity regions in system behaviour.

We propose a dynamic model that predicts MapReduce cluster performance, in our case the average service time, based on the number of nodes and the number of clients. To the best of our knowledge this is the *first dynamic performance model for MapReduce systems*.

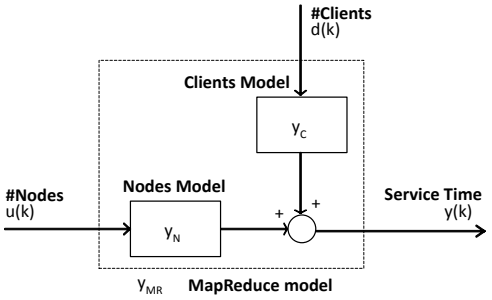


Fig. 5. MapReduce control theoretical model: y_{MR}

The structure of our model can be seen in Figure 5. Our control input $u(k)$ is the number of nodes in the cluster while the changes in clients $d(k)$ is considered as a measurable disturbance. Our output $y(k)$ is the average service time of a job in the k^{th} time interval. The complete list of modelling notations used to define the model structure is given in Table 1

As in the operating region our system is linear we can apply the principle of superposition to calculate the output:

$$y(k) = y_C \cdot d(k) + y_N \cdot u(k) \quad (2)$$

where y_N is the discrete time model between service time and the number of nodes and y_C is the discrete time model between service time and the number of clients.

y	System output: average client request service time.
u	Control input: number of processing nodes in cluster
d	Disturbance input: number of clients
τ	The time delay after which the effect of an input change is visible on the output.
y_N	Nodes model - captures the effect of node variations on service time.
a_i, b_j	Parameters of the nodes model
y_C	Clients model - captures the effect of client variations on service time.
c_p, l_r	Parameters of the clients model
n_C, n_N	Number of past output values affecting current output.
m_C, m_N	Number of past input values affecting current output
y_{MR}	MapReduce model

TABLE 1
Definition of modelling notations.

3.5 Identifying model parameters

The identification procedure is fairly simple and can be easily automated. Its methodology is summarized in Figure 6.

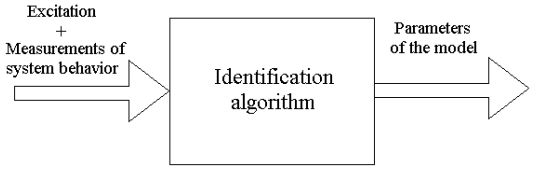


Fig. 6. General identification procedure

The identification algorithm is chosen from the wide literature of identification algorithms (see [33] for a review) used in control theory. In our case we chose the prediction error estimation algorithm from [34]. This method is a classical method in control theory and has the advantage of putting an emphasis on the model accuracy in predicting the next observation rather than on its difference from a corresponding statistical model, as it is the case for least square and maximum likelihood identification methods. The method first provides a continuous time model which is further on discretized using Tustin bilinear transformation [20].

After the choice of the algorithm, the user should provide the excitation for the system (the MapReduce system in our case) and the measurements of the system's response to this excitation. The most simple excitation has the shape of a step, where the excitation signal has a single step increase and then it remains constant for the rest of the experiment.

After providing the excitation and the system's response as the input values of the identification algorithm, the user gets the parameters of the mathematical model of the system as the algorithm output.

As our system has large time constants (>300s as it can be seen form Figures 3 or 4) we determine that a sampling period of 30 seconds is sufficient. The reasoning behind this is based on Shannon's law for choosing the sampling period (for more information, see [20]).

3.5.1 System identification without disturbance

For the system without disturbance the identification is done by analysing the system behaviour from Figure 3. Furthermore, a step in the number of nodes is used to identify the model between the service time and the number of nodes. The identified, disturbance free, system model will have the following form:

$$y_{N(k)} = \sum_{i=1}^{n_N} a_i \cdot y_{N(k-i)} + \sum_{j=0}^{m_N} b_j \cdot u_{(k-\tau_N-j)} \quad (3)$$

where the coefficients a_i and b_j along with the deadtime τ_N are to be found by the identification algorithm. Furthermore, y_N models the effect of nodes changes on service time and u is the number of nodes. The parameters a_i capture the effect of past outputs on current outputs, while b_j capture the correlation between current outputs and past cluster sizes.

3.5.2 Disturbance model identification

For the changes in the number of clients the system behaviour is presented in Figure 4. The same strategy of a step increase, in this case in the number of clients, will be used to identify the model between the service time and the number of clients. The model has the following form :

$$y_{C(k)} = \sum_{p=1}^{n_C} c_p \cdot y_{C(k-p)} + \sum_{r=0}^{m_C} l_r \cdot d_{(k-\tau_C-r)} \quad (4)$$

where the coefficients c_p and d_r along with the delay τ_C are to be found by the identification algorithm. Furthermore, y_C models the effect of client changes on service time and d is the number of nodes.

The given equations (3) and (4) can be applied to other systems straight forward, with the re-identification of the equation parameters. The models identified are valid for a client mix where the variance in the mean response time of the clients requests is 25%. In our case we have 5 different requests running. Moreover contrary to existing solutions that build a model for each job, *our model covers an infinite set of jobs, the only condition being that their response time variance is within the specified limit.*

All the variables used in this section are deterministic variables. Their values depend upon the dynamics of system on which the identification algorithm has been run. In our case the exact, numerical values of the variables are given further on in Section 5, in equations (5) and (6).

3.6 General algorithm for building dynamic performance models of Big Data systems

A typical system design time question that might arise is that, given a budged (that translates directly to number of nodes), what is the maximum amount of clients that our service can serve? Furthermore, to minimise cost, one would like to maximise the resource usage meanwhile, making sure that client request run as fast as possible? Based on our experiences with modelling MapReduce systems, we propose the following general algorithm, that answers the previous question, and which can be used to find the linearisation point around which a dynamic model can be identified:

1) Choose the number of resources desired for nominal operating conditions based on financial constraints.
2) Increase the number of clients until system throughput starts to saturate.
3) Set this point of saturation defined by $(\#resources, \#clients)$ as your set point for linearisation. Here the system is fully utilised. Adding more clients would decrease performance, adding more resources would not improve performance.
4) Identify a dynamic performance model around this operation point using the identification procedure presented in Section 3.5.

The main advantage of this algorithm is that we will have a model of our system at the edge of full utilisation, which is where we normally want our steady state of the system to be, to minimise costs. The model can predict what happens when more clients arrive at full utilisation and can help a controller decide upon the number of resources to add to reach once again full utilisation and keep the desired performance levels.

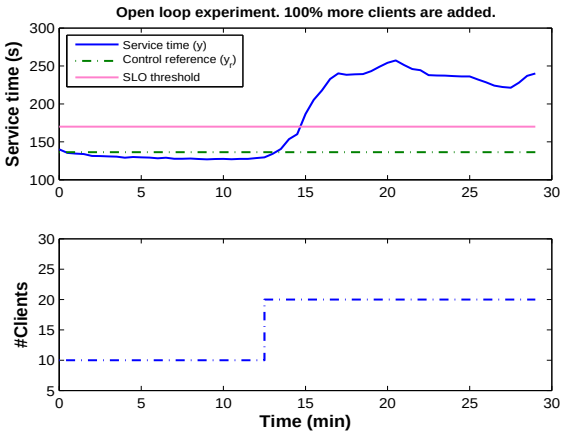
However, the model is valid around an operating point. As a consequence, if we model our system using 10 clients and 20 nodes and then test our control with 100 clients and 200 nodes the model might loose validity, depending on system linearity. However, this can be easily addressed by re-triggering the automatic identification procedure, presented above in Section 3.5.1, when we leave the operating region. Moreover, the complexity of the parameter identification algorithm remains unchanged, as the identification algorithm itself works independently from actual value of the number of clients or cluster size. As it focuses on the capturing the effect of a change in the client or node count from current value.

4 CONTROL

4.1 Control motivation

Before presenting the control architecture let us see what happens when we have no automatic control but only a sudden step increase in the number of clients (see Figure 7). Such a bursty increase in the number of clients occurs frequently in practice, see [35] who analyses a 10-month log production of Yahoo's supercomputing cluster. We call this experiment an open loop experiment since the feedback loop from the output to the input does not exist in this case.

In our case we can see that, when 100% more clients are added, the systems service time quickly exceeds the reference threshold defined in the Service Level Agreement (SLA). This behaviour is not desirable since it automatically implies losses.



E

Fig. 7. Baseline experiment - no control, #Nodes=20

4.2 Control architecture

When it comes to control the MapReduce system, we can distinguish two separate cases. First, we have the relaxed performance - minimal resources case where the service provider needs to keep the system service time below the reference threshold, defined in the SLA, but also wants to minimize the number of used resources (in this case the number of system nodes) to reduce cost. Therefore, if this is specified in the SLA, the client accepts that *for a small amount of time* the service time could exceed the reference threshold. The second case is the strict performance one, when the service provider has a *very strict demand* from the client in keeping the service time below the reference all the time. This can be the case for online brokerage industry where the service unavailability costs about 6.48 million dollars per hour [16]. Since the number of clients trying to use the service is unpredictable, the service provider is accepting a considerable increase in the number of the system nodes (therefore a increase in the utilization cost) in order to respect the SLA and face the client increase.

The complete schema of our control architecture, which comes to address these two challenges, is presented in Figure 8. The variables used in the figure are defined in Table 2. As in Figure 5, we consider the MapReduce system having two inputs: the control input $u(k)$ which is the number of nodes in the cluster and the exogenous input $d(k)$ which is number of clients to connect to the system, and one output which is the service time $y(k)$.

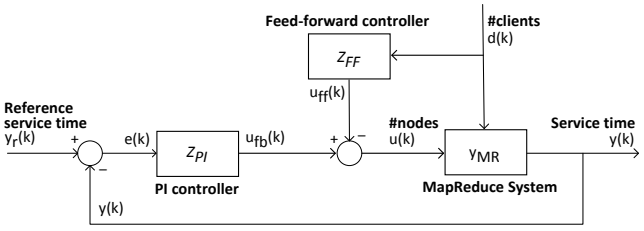


Fig. 8. MapReduce Control architecture

y_r	Reference average service time set in the SLA.
Z_{PI}	Discrete time PI feedback controller.
u_{fb}	Control output of the PI controller.
K_p, K_i	Feedback control tuning parameters.
e	Difference between measured and desired service times.
e_{lim}	Event threshold on the error. If the threshold is exceeded actuation is necessary.
T_{act}	Time elapsed since the last actuation.
Z_{FF}	Discrete time feedforward controller.
u_{ff}	Control output of the feedforward controller.

TABLE 2
Definition of control variables.

Our answer to the challenges of the first use case is the PI feedback controller Z_{PI}. When we have strict SLA constraints we add the feedforward controller Z_{FF}. The size of the workload in this case is assimilated to an exogenous disturbance input. However, as we can accurately measure it on-line the feedforward controller can use this information in order to assure a faster controller response. This is done by counteracting the disturbance before its effects can be measured on the output. These type of control architectures have well proven their efficiency in many different engineering domains, from the early 1900's.

As we can see from Figure 8 the PI feedback controller Z_{PI} periodically monitors the service time and reacts to difference between the measured $y_r(k)$ and reference $y_r(k)$ values. While, the feedforward controller Z_{FF} monitors and reacts to client variations. In the next sections we discuss these different control strategies in detail.

5 Model Validation

The identification procedure from Section 3.5 is used to find the model of the MapReduce System (y_{MR}) composed of two submodels y_N and y_C (see equation (2)). As stated before in Section 3.5.1 the prediction error algorithm is implemented to find the parameters of these functions.

5.1 System identification without disturbance

The identified model for the node changes can be seen in Figure 9.

A step in the number of nodes is used to identify the model between the service time and the number of nodes. As it can be seen, the model found by the algorithm captures well the system dynamics with a fit level of 86.53%.

Four our case, the form of equation (3) and the values of the coefficients a_i and b_j, are given in equation (5). As it can be seen, the identified delay τ_N is 5 sampling periods.

$$y_{N(k)} = 0.919 \cdot y_{N(k-1)} - 0.179 \cdot u_{(k-5)} - 0.179 \cdot u_{(k-6)} \quad (5)$$

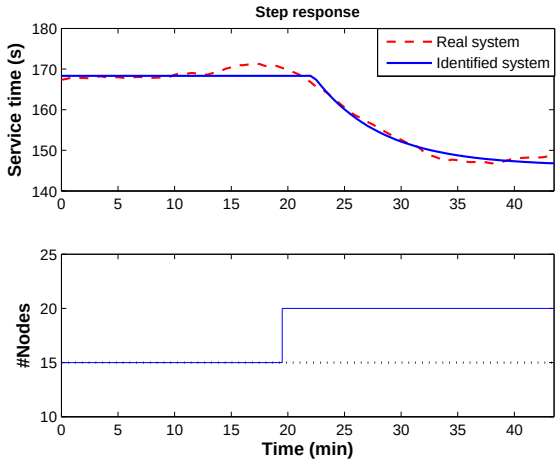


Fig. 9. Identification of the undisturbed system. It predicts the effect of nodes changes on job runtime. #Clients=10

5.2 Disturbance model identification

Figure 10 shows the step responses for the identified and measured systems, in the case of a change in the number of clients. As we can see, the identified model also follows closely the measurements taken from the real system, presenting a 87.94% fit. With the coefficients c_p and d_r found by the algorithm, equation (4) becomes (6), which captures the effects of client variations on service time:

$$y_{C(k)} = 0.7915 \cdot y_{C(k-1)} + 1.0716 \cdot d_{(k-8)} + 1.0716 \cdot d_{(k-9)} \quad (6)$$

It can be seen that for this model, the delay is different than the previous one, τ_C being equal to 7 sampling periods.

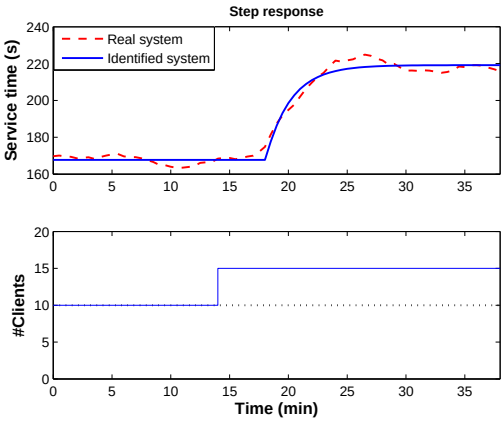


Fig. 10. Identification of the disturbance model. It captures the effect of the changes in the number of clients on job runtime. #Nodes=20

6 Relaxed Performance - Minimal Resource Control

It is well proven in the control theory literature, see for example [36] among many others, that in cases, where we have a first order model structure with an actuation delay, a Proportional Integrator (PI) feedback controller is sufficient to control the system, even if in reality the system itself has a more complex dynamics and only the general tendency of the system is captured by the first order model. Therefore, since from equations (3) and (4) we notice a first order system model with input delay, a PI feedback strategy is designed.

6.1 PI Feedback Control

In the first case we consider a simplified version of the control architecture from Figure 8, where only the PI feedback controller is running and the feedforward controller is not active ($Z_{FF} = 0$). The standard equation of a sampled time PI controller is very well known in the control theory literature (see [20]) and is given by equation (7):

$$u_{fb_{(k)}} = u_{fb_{(k-1)}} + (K_p + K_i)e_{(k)} + K_i e_{(k-1)} \quad (7)$$

The controllers parameters are determined to assure stability, when we close the loop with the controller, and no overshoot. In control theory we talk of overshoot when the system output exceeds its target before stabilizing on a reference value. Furthermore, as we would like to avoid a highly aggressive controller, the control response to the disturbance is somewhat slow. The reason behind this is the minimization of the number of changes in the number of nodes, because of financial and energetic constraints. Based on these requirements and the identified model given in equation (5), we mathematically computed the value of $K_p = 0.0012372$ and $K_i = 0.25584$ for our controller.

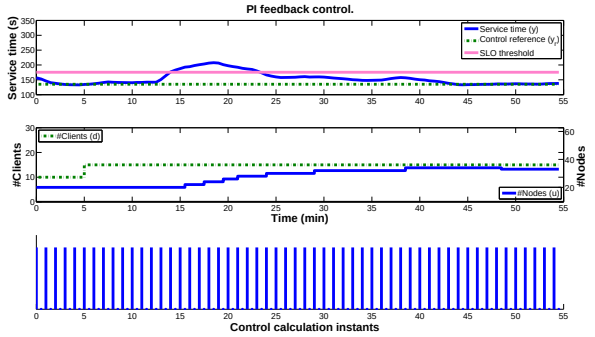


Fig. 11. Closed loop experiments - Feedback Control

The results are given in Figure 11 which shows our controllers response to a 50% change in the number of clients. We can see that as the controller is determined to have a slow settling time, the SLA threshold is breached for a short amount of time but the controller will always take the service time to our reference value. The controller steadily increases the number of nodes until service time recovers. It can also be seen that the number of nodes to be added to the system to keep the SLA is recomputed at each sampling interval (see the 3^{rd} plot)

and therefore is changing frequently (see the 2^{nd} plot). The control automatically computes the number of nodes to add at each time and this number is not necessarily the same. Since frequent cluster reconfiguration are not desired, in Section 6.2 below, we further develop the previous algorithm to reduce the number of node changes, while also keeping the SLA.

6.2 Event based PI control - Minimizing the number of actuations

The field of control in Big Data cloud environments brings us specific control constraints. The adding and removing of resources takes considerable time and has energetic and financial costs. We therefore want to avoid as much as possible such quick changes in the control signal. One approach to minimize the number of actuations can be found in the event-based control theory.

Event-based controllers have emerged recently as a viable alternative to periodic controllers when it comes to handling constraints on the number of actuations, limited communication or computation bandwidth, constraints on power consumption, etc [37], [38]. The basic idea behind this theory is that we don't need to calculate a new control value with every new measurement, instead the control is calculated only when the system output changes more than a certain threshold, since the last actuation. This concept is illustrated in Figure 12 where a simple example is shown to clarify the difference between the control instants of time-based and event-based controllers. For a more detailed view of event-based control theory see [39], [37], [38].

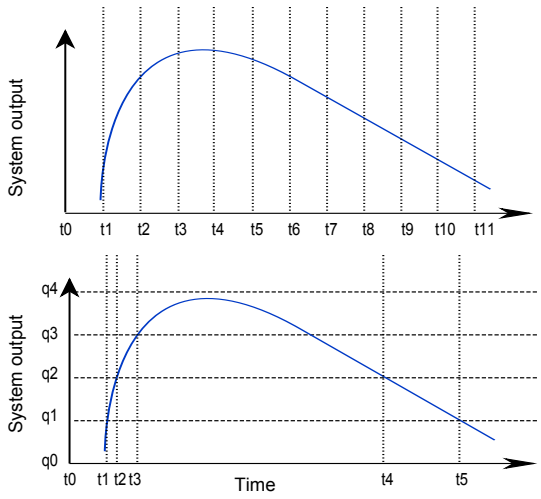


Fig. 12. Difference between time based and event-based control instants.

Figure 12 shows us that, while the control instants of time-based control follow a fixed time period $t_{i+1} - t_i$, for the event-based case the control is calculated only when the system output changes more than a fixed threshold value e_{lim}. The Y axis in this simple case can be any feedback signal used for control purposes.

For the case of controlling cloud resources we see the main advantage of using event based controllers in minimizing the number of actuations. This minimization can be done using the extra tuning parameter introduced by the event based controller, namely the error threshold limit e_{lim}. This value can be used to minimize the actuation count while still retaining acceptable performance. The PI control architecture using the event-based approach can be seen in Figure 13.

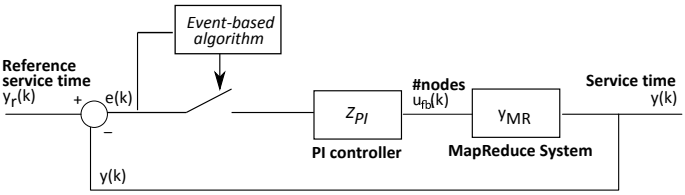


Fig. 13. Event-based PI feedback Control Architecture

We can see that the event-based module functions act as a switch in the system that, when activated allows for a new control value to be calculated. In the case of our implementation the event-based algorithm is the following: a new control value is calculated only if the difference between the current error and last error value for which control was calculated is greater than this threshold e_{lim}. The detailed description of the structure for the event based PI controller is in [40]. Starting from their results, we use an exponential forgetting factor to limit the impact of the integrator component, after periods of long inactivity, on the control signal. Finally our event-based PI controller developed is governed by equation (8):

$$\begin{aligned} u_{pi} &= u_p + u_i, where \\ u_p &= K_p \cdot e \\ u_i &= u_{i-1} + K_i \cdot f(e, e_{lim}, T_{act}); \end{aligned} \tag{8}$$

where e is the error and e_{lim} is the event threshold. As soon as the error becomes larger that this threshold, the control value is updated. u_p and u_i are respectively the proportional and integral terms of the control. K_p and K_i are respectively the proportional and integral coefficient of the PI controller. f is a function that gathers a forgetting strategy that the event-based implementation requires (see [40] for further informations). Finally, T_{act} denotes the time elapsed since the last update of the control value. The control parameters for the event based PI are the same as the ones calculated for the fixed sampled PI controller. The parameter e_{lim} is found as the maximum value for which performance is still acceptable. The results of implementing the Event Based PI controller on the same system as previously described are given in Figure 14.

We see that the event based PI controller also manages to keep the service time below the threshold in the presence of perturbation just like the classical PI controller.

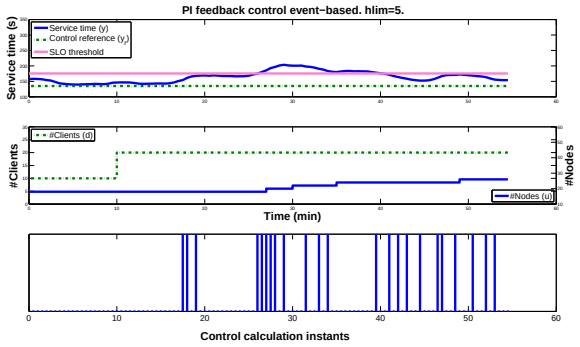


Fig. 14. Closed loop experiments - Event Based PI Control

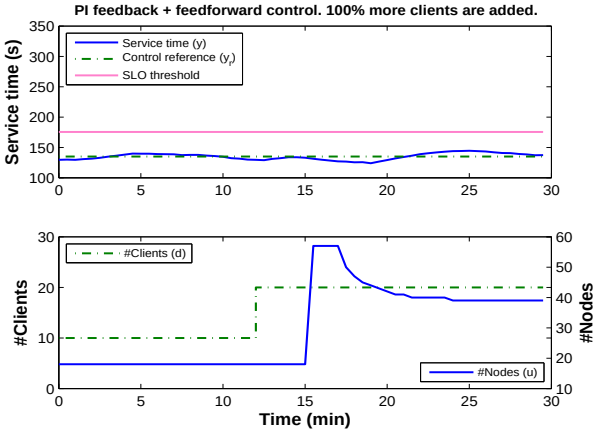


Fig. 15. Closed loop experiments - Feedback and Feedforward Control

One can also see that, when comparing Figure 11 with Figure 14, the control signal to be applied is computed only several times and the amount of switches in the nodes number is therefore drastically diminished as well. We have reduced by half the changes in system nodes (4 changes with the event based controller compared to 8 without). Although the performance is just a bit worse, its benefits are that it brings less energetic cost for the cloud provider and less financial cost for the user.

7 Strict performance - Feedforward Control

To satisfy strict performance constraints a fast feedforward control strategy is designed, to pro-actively reject the disturbance before its effect is observed on the output. In our case the disturbance represents a change in the number of concurrent clients. The purpose of this controller is to create a control signal that, once it has passed through the plant, cancels out the effect of the disturbance. If the model is 100% accurate the net effect on the service time should be zero, but because of the inherent model uncertainties this is never the case in practice. Our controller is determined using the standard feedforward formula

$$Z_{ff}(z) = -y_N(z)^{-1} y_C(z)$$

where Z_{ff} is the discrete time feedforward controller and y_N, y_C are the discrete time models from Figure 5. The equation of the computed feedforward controller is given in equation (9):

$$u_{ff_{(k)}} = 0.791 \cdot u_{ff_{(k-1)}} + 5.97 \cdot d_{(k-2)} - 5.486 \cdot d_{(k-3)} \quad (9)$$

The effect of adding the feedforward control to the already existent feedback controller can be seen in Figure 15. One can observe that although so far we have tested our controllers with a jump of 50% more clients, here we test our control with worst conditions, a jump of 100% more clients, to highlight the effectiveness of feedforward control in comparison to just feedback control. While the feedback control shortly breached the SLO even with a 50% increase, by adding the feedforward component we can see that the controller response is increased and manages to keep the response time below the SLA threshold all the time even if we double the current workload. Furthermore, the feedback term compensates for all the model uncertainties that were not considered when calculating the feedforward one and assures that the steady state error converges to 0. This comes though with a cost, the increase in the number of nodes that are utilized. The number of nodes to be added is also recomputed at each sampling interval and therefore it is not plotted again.

8 Related work

In this section we overview the different approaches to model and control the performance of MapReduce systems.

8.1 Mapreduce performance modelling

Many studies have been already performed on how to model the performance of the MapReduce framework. These can be grouped together into the following categories. Analytical or first principle models are detailed MapReduce models that capture the inner workings of the different phases of a classical Hadoop MapReduce job execution flow, see [41], [42], [43]. Vianna et al. [44] propose a hierarchical model that combines a precedence graph with a queueing network to model the intra-job synchronisation constraints. Some as Jockey [45] use a simulator that captures the complex interdependencies of a job and makes use of previous runtime statistics to predict job runtime. On the opposite side there are the regression and black box models. These are coarse grained models that don't try to capture the specificities of the MapReduce framework but instead build upon job profiling, namely predicting the response time of future jobs based on past experience or exploratory runs. In the latter case the model parameters are generally found by

running the job on smaller set of the input data and using regression techniques to identify model parameters. The differences between these regressive approaches lies mostly in the components used to set up the regressive model. Some authors develop statistical models made of several performance invariants such as the average, maximum and minimum run times of the different job cycles [27], [26]. As most MapReduce jobs are batch jobs that are run frequently, some propose building a profile database [27] to predict job runtime. Others employ a static linear model that captures the relationship between job runtime, input data size and the resources allocated for the job [28]. Furthermore, there are those who analyse long term traces to classify jobs into several runtime categories, for example from a 10-months logs of Yahoo's M45 supercomputing clusters running MapReduce. They use two separate algorithms for the prediction of service completion times: a distance-weighted average algorithm and a locally-weighted linear regression method. The linear regression based method was proven to scale better for varying input sizes, see Kavulya [35]. Primary Component Analysis has been also used to determine the MapReduce/Hadoop components that have the biggest influence on performance of MapReduce jobs [32]. This approach mixes the non-application specific Hadoop configuration parameters with the statistical averages collected from the traces of previous job runs. They find that as different applications have varying CPU, network bandwidth, data storage requirements, the use clustering analysis is advised to group jobs and build a separate model for every group to achieve better model performances.

However, all the presented models are job level models and therefore cannot capture the effects of workload variations in a MapReduce cluster. Furthermore, all these models are static models and don't capture the dynamics of a MapReduce system, namely what happens during a workload change until the system reaches its new steady state. Meanwhile, the many years of experience in building control algorithms for physical systems has shown that capturing the dynamics of the system is crucial to know when and how to control.

8.2 Improving the MapReduce Framework

There exist many attempts to improve upon MapReduce performance either through framework modifications or by optimizing the framework parameters. Sailfish [46] is a new MapReduce framework that, by aggregating intermediate data, improves performance by batching disk I/O. Hadoop++ [47] improves job performance for analytical queries using a new non-invasive indexing technique. Yarn [48] brings several performance improvements while supporting additional processing models. However, non of these framework provides any control mechanism than can guarantee performance in face of a varying workload. Furthermore, with the advent of cloud solution, there are many projects on improving MapReduce performance in the cloud. Spark [19], for example, generalises the MapReduce model and can deal with new workload such as streaming, iterative algorithms and interactive queries. Although it is not yet as mature as Hadoop it has been shown to outperform Hadoop by a couple of orders of magnitude in many cases. AsterixDB [17] is a new Big Data Management System that stores, indexes and manages semi-structured data. Because of its knowledge of data partitioning and indexing it can avoid to always scan data to process queries. Stratosphere [18] further extends the MapReduce model, allowing for more operators than just map and reduce and does much better on iterative algorithms than traditional Hadoop. Furthermore, due the generality of the algorithms developed in this paper, they can be applied to any of the previously listed frameworks to guarantee performance requirements.

8.3 Guaranteeing MapReduce Performance

By guaranteeing MapReduce performance we think of the on-line adaptation of frameworks resources or any of its parameters to achieve the required job deadlines. For example SteamEngine [49] introduces an on-line performance and energy optimization algorithm for MapReduce applications running on virtualised clusters, such as Amazon EC2. It makes use of both off-line and on-line job profiling to predict job finish times. The performance optimization is done by regularly predicting the job finish time and using a simple heuristic to control the amount of resources available for tasks. Namely, if the predicted finish time, at any time of the job life-cycle, is more than the expected finish time then the algorithm increases the amount of resources (adds more nodes) through cluster scaling. The cluster scaling optimization is done only in the map phase, and the earlier it's done the better is the improvement. Verma [50] proposes ARIA, an automatic resource inference and allocation engine for MapReduce. ARIA can, at run time, allocate the appropriate resources (slots) to a job so that the job meets its time constraints. Jockey [45] monitors job performance and dynamically adjusts its resources to maximise economic utility, while minimising its impact on the rest of the cluster. While all the previous approaches propose fine grained job level performance control at a scheduler level, we propose to add course grained control by controlling the average performance of a group of jobs in the cluster. Furthermore, while these methods require modifying the schedulers and algorithms deployed by the MapReduce cluster, our control architecture is non-invasive and can be used in parallel with any of the previous listed scheduling algorithms. Moreover our algorithm can be easily automated to be used by an average user with their distribution, without an in depth knowledge of the inner workings of the MapReduce framework. While the fine grained scheduling techniques optimise the resource usage of the current resources, our course grained technique can handle workload spikes and fluctuations.

9 Conclusions and Future Work

This paper presents the design, implementation and evaluation of the first algorithm for creating dynamic performance models for Big Data MapReduce systems. Moreover we identify two major performance constraint use cases: relaxed - performance, minimal resource and strict performance constraints. For the first case we develop and implement a PI feedback control mechanism. To further minimize the number of control actuations, an event-based feedback controller is introduced as well. With the latter, the changes in the number of nodes is diminished considerably, while the performance is kept at almost the same level. For the second case we develop and implement a feedforward controller that efficiently suppresses the effects of large workload size variations. All the control algorithms are validated online on a real 60 node MapReduce cluster, running a data intensive Business Intelligence workload. Our experiments show that the controllers are successful in keeping the performance constraints set in the service level agreement.
Further investigations are necessary in some areas and are studied now, such as:

1) implementing the control framework in an on-line cloud such as Amazon EC2.
2) improve upon the strict performance constrained control with event-based techniques.
3) develop an on-line identification mechanism
4) add other metrics to our model such as throughput, availability, reliability.

Acknowledgments

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

This work has been supported, in part, by the European FP7 research project AMADEOS Grant Agreement 610535 on Systems of Systems.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

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] Y. Shen, *Enabling the New Era of Cloud Computing: Data Security, Transfer, and Management*, 1st ed. Hershey, PA, USA: IGI Global, 2013.
[3] "Amazon emr," http://aws.amazon.com/elasticmapreduce/.
[4] "Hd insight," https://azure.microsoft.com/hdinsight/.
[5] A. G. García, I. B. Espert, and V. H. García, "Sla-driven dynamic cloud resource management," *Future Generation Computer Systems*, vol. 31, pp. 1–11, 2014.
[6] E. Casalicchio and L. Silvestri, "Mechanisms for sla provisioning in cloud-based service providers," *Computer Networks*, vol. 57, no. 3, pp. 795–810, 2013.
[7] D. Serrano, S. Bouchenak, Y. Kouki, F. A. de Oliveira Jr., T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens, "{SLA} guarantees for cloud services," *Future Generation Computer Systems*, no. 0, pp. –, 2015.
[8] Project, "Mycloud," http://mycloud.inrialpes.fr/.
[9] ——, "Harness," http://www.harness-project.eu/.
[10] C. Tian, H. Zhou, Y. He, and L. Zha, "A dynamic MapReduce scheduler for heterogeneous workloads," in *Proceedings of the 8th International Conference on Grid and Cooperative Computing (GCC)*, Washington, DC, USA, 27-29 Aug. 2009, pp. 218–224.
[11] A. Sangroya, D. Serrano, and S. Bouchenak, "Benchmarking Dependability of MapReduce Systems," in *IEEE 31st Symposium on Reliable Distributed Systems (SRDS)*, Irvine, CA, 8-11 Oct. 2012, pp. 21 – 30.
[12] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX Conference on Operating systems design and implementation (OSDI)*, Berkeley, CA, USA, 8-10 Dec. 2008, pp. 29–42.
[13] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou, "Workload characterization on a production Hadoop cluster: A case study on Taobao," in *IEEE International Symposium on Workload Characterization (IISWC)*, La Jolla, CA, 4-6 Nov 2012, pp. 3–13.
[14] Y. Chen, S. Alspaugh, and R. H. Katz, "Design insights for MapReduce from diverse production workloads," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-17, Jan 2012.
[15] H. Jin, X. Yang, X.-H. Sun, and I. Raicu, "Adapt: Availability-aware mapreduce data placement for non-dedicated distributed computing." in *ICDCS*. IEEE, 2012, pp. 516–525.
[16] Evolven. (2013, 18 September) Downtime, outages and failures - understanding their true costs. http://www.evolven.com/blog/downtime-outages-and-failures-understanding-their-true-costs.html.
[17] A. Behm, V. R. Borkar, M. J. Carey, R. Grover, C. Li, N. Onose, R. Vernica, A. Deutsch, Y. Papakonstantinou, and V. J. Tsotras, "Asterix: towards a scalable, semistructured data platform for evolving-world models," *Distributed and Parallel Databases*, vol. 29, no. 3, pp. 185–216, 2011.
[18] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke, "Nephele/pacts: a programming model and execution framework for web-scale analytical processing," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 119–130.
[19] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10–10.
[20] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback control of computing systems*. John Wiley & Sons, Inc., New Jersey, 2004.
[21] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A systematic survey on the design of self-adaptive software systems using control engineering approaches," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, June 2012, pp. 33–42.
[22] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, V. Papadopoulos, Alessandro, S. Ray, M. Sharifloo, Amir, S. Shevtsov, M. Ujma, and T. Vogel, "Software Engineering Meets Control Theory," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Firenze, Italy, May 2015. [Online]. Available: https://hal.inria.fr/hal-01119461
[23] L. Malrait, N. Marchand, and S. Bouchenak, "Modeling and control of server systems: Application to database systems," in *Proceedings of the European Control Conference (ECC)*, Budapest, Hungary, August 2326 2009, pp. 2960–2965.
[24] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th International Conference on Autonomic Computing*, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 1–10.
[25] T. White, *Hadoop: the definitive guide*. O'Reilly Media, CA, 2012.
[26] L. Xu, "MapReduce framework optimization via performance modeling," in *IEEE 26 International Parallel & Distributed Processing Symposium (IPDPS)*, Shanghai, China, 21-25 May 2012, pp. 2506–2509.
[27] A. Verma, L. Cherkasova, and R. Campbell, "Resource provisioning framework for MapReduce jobs with performance goals," in *Middleware 2011*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 7049, pp. 165–186.

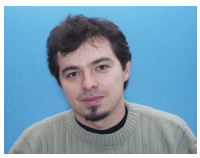
[28] F. Tian and K. Chen, "Towards optimal resource provisioning for running MapReduce programs in public clouds," in *IEEE International Conference on Cloud Computing (CLOUD)*, Washington, DC, USA, 4-9 July 2011, pp. 155–162.
[29] M. J. Carey, "Bdms performance evaluation: Practices, pitfalls, and possibilities," in *Selected Topics in Performance Evaluation and Benchmarking*. Springer, 2013, pp. 108–123.
[30] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard, "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, Washington, DC, USA, 2005, pp. 99–106.
[31] Hadoop, "Hadoop 1.1.2 release notes," http://hadoop.apache.org/docs/r1.1.2/releasenotes.html.
[32] H. Yang, Z. Luan, W. Li, and D. Qian, "MapReduce workload modeling with statistical approach," *Journal of Grid Computing*, vol. 10, pp. 279–310, 2012.
[33] L. Ljung, *System Identification*. Wiley Encyclopedia of Electrical and Electronics Engineering, 1999.
[34] T. Söderström and P. Stoica, *System identification*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1988.
[35] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production MapReduce cluster," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*, Washington, DC, USA, 2010, pp. 94–103.
[36] S. Guillermo J, *PID Controllers for Time-Delay Systems*. Birkhauser Boston, 2005.
[37] A. Seuret, C. Prieur, and N. Marchand, "Stability of nonlinear systems by means of event-triggered sampling algorithms," *Journal of Mathematical Control and Information*, 2013.
[38] N. Marchand, S. Durand, and J. F. Guerrero-Castellanos, "A general formula for event-based stabilization of nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 5, pp. 1332–1337, 2013. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00675946/
[39] K. J. Aström, "Event based control," in *Analysis and Design of Nonlinear Control Systems*, A. Astolfi and L. Marconi, Eds. Springer Berlin Heidelberg, 2008, pp. 127–147.
[40] S. Durand and N. Marchand, "Further results on event-based pid controller," in *Proceedings of the European Control Conference (ECC)*, 2009.
[41] H. Herodotou, "Hadoop performance models," *CoRR*, vol. abs/1106.0940, 2011. [Online]. Available: http://arxiv.org/abs/1106.0940
[42] X. Lin, Z. Meng, C. Xu, and M. Wang, "A practical performance model for hadoop mapreduce," in *Cluster Computing Workshops, 2012 IEEE International Conference on*. IEEE, 2012, pp. 231–239.
[43] X. Yang and J. Sun, "An analytical performance model of mapreduce," in *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, 2011, pp. 306–310.
[44] E. Vianna, G. Comarela, T. Pontes, J. Almeida, V. Almeida, K. Wilkinson, H. Kuno, and U. Dayal, "Analytical performance models for mapreduce workloads," *International Journal of Parallel Programming*, vol. 41, no. 4, pp. 495–525, 2013.
[45] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: Guaranteed job latency in data parallel clusters," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 99–112. [Online]. Available: http://doi.acm.org/10.1145/2168836.2168847
[46] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves, "Sailfish: A framework for large scale data processing," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 4:1–4:14.
[47] J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, "Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing)," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 515–529, Sep. 2010.
[48] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16.
[49] M. Cardosa, P. Narang, A. Chandra, H. Pucha, and A. Singh, "STEAMEngine: Driving MapReduce provisioning in the cloud," in *18th International Conference on High Performance Computing (HiPC)*, Bengalore, India, 18-21 Dec. 2011, pp. 1–10.
[50] A. Verma, L. Cherkasova, and R. H. Campbell, "Aria: automatic resource inference and allocation for mapreduce environments," in *Proceedings of the 8th ACM international conference on Autonomic computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 235–244.

Mihaly Berekmeri received his PhD in Automatics from the University of Grenoble, France in 2015. He is member of the SYSCO and LIRIS reseach groups at Gipsa-Lab and INSA-Lyon. His research focuses on the development of control theoretical tools for Big Data, distributed computing systems and cloud services.

Damian Serrano is a post-doctoral researcher in the IRISA research group, part of the University of Rennes 1, France. He received his PhD degree in computer science from the Universidad Politecnica de Madrid in 2010. He spent two years as a post-doctoral researcher in the SARDES project, part of Inria and LIG. His research focuses in designing and developing cloud services that are able to provide Quality-of-Service (QoS) guarantees.

Sara Bouchenak is Professor in Computer Science at INSA-Lyon. She is a member of ACM, IEEE, and EuroSys; she was an officer of the French chapter of ACM-SIGOPS. She was a visiting professor at Universidad Politecnica de Madrid, Spain, in 2009/2010, and an associate researcher at EPFL, Switzerland, in 2003. She received her PhD in computer science from Grenoble Institute of Technology in 2001. She is a member of the LIRIS research group, where she conducts research on highly-available, dependable and manageable distributed computer systems.

Nicolas Marchand is a researcher at CNRS, France since 2002. Head of Control Systems Department, Deputy director of GIPSA-lab since 2011. Prior to this he spent two years as Assistant professor at the University Paris-Sud Orsay (2000) and one year as researcher (ATER) at the technological university institute of Villeurbanne(1999), France. He received his PhD in control from the Grenoble Institute of Technology in 1999. His research interests focus on the control of computing systems and control and stabilization of biomimetic robots.



Bogdan Robu is a Professor at the University of Grenoble, France and a researcher in the GIPSA-lab laboratory in Grenoble since 2012. Prior to this he had a one year teaching position at Toulouse University, France. His research interests focus on the control of computing systems in general,with more focus on Cloud Control and Transactional Memory control, robust control and PDE modeling. He received his PhD in control from the Toulouse University in 2010.