2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

BBQ: Elastic MapReduce over Cloud Platforms

Nikolaos Chalvantzis, Ioannis Konstantinou and Nectarios Koziris
CSLAB, National Technical University of Athens
{nchalv,ikons,nkoziris}@cslab.ece.ntua.gr

***Abstract*—Cloud infrastructure services such as Amazon EMR allow users to have access to tailor-made Big Data processing clusters within a few clicks from their web browser, thanks to the elastic property of the cloud. In virtual cloud environments, resource management is desired to be performed in a way which optimizes utilization, thus maximizing the value of the resources acquired. As cloud infrastructures become increasingly popular for Big Data analysis, the execution of programs with respect to user selected performance goals, such as job completion deadlines, remains a challenge. In this work we present BARBECUE (joB AwaRe Big-data Elasticity CloUd managEment System), a system that allows a Hadoop MapReduce virtual cluster to automatically adjust its size to the workload it is required to execute in order to respect individual jobs' completion deadlines without acquiring more resources than the least necessary. To that end, BBQ's Decision Making module uses a Performance Model for MapReduce jobs which can express cluster resources (i.e., YARN Container capacity) and execution time as a function of the number of nodes in the cluster. BBQ leverages the abstraction of YARN, making it feasible for integration with other execution frameworks, such as Spark, with the necessary changes to its pluggable Decision Making module. We also add a new feature to Hadoop MapReduce which can now dynamically, on-the-fly update the number of selected ReduceTasks in cases where the cluster is expanded, so that our system makes full use of the resources it has acquired during the reduce phase of the execution. BBQ uses an adaptation of the *hill climbing* algorithm to estimate the optimal combination of *number of nodes* and *reduce waves* given a known job, its data input and an execution deadline. The attendees will be able to watch the system perform cluster resizes in real-time in order to execute its assigned jobs in time.**

I. Introduction

Cloud computing [14] becoming a mainstream trend about a decade ago, changed the way big data is processed and stored. Having access to unlimited computing power and storage resources, users and organizations are now able to respond to the ever growing need of processing huge amounts of data. According to IBM, approximately 2.5 quintillion bytes are created every day [2]. This increasing data production enforces the use of cloud computing as a popular practice but also brings new challenges, concerning the way cloud cluster infrastructures are being managed and provisioned. Virtual cloud environments are an agile solution for Big Data analysis systems thanks to elasticity [5], [9]. The elastic property of the cloud allows administrators to manage the computing resources available to them according to their needs with minimal effort. Resource management in virtual cloud environments is an area with a lot of research potential, since the line between good and bad practices is extremely thin. Currently, many users handle their clusters' provisioning needs by using rules of thumb and personal experience with specific frameworks. As big data analysis solidifies its status as an indispensable tool for cross-field academic as well as innovative entrepreneurial research and cloud services accessible to a larger non-expert user base, there can be cases where users lack the background to make cluster provisioning decisions. Recent research projects have tried to explore the factors which influence such complex problems and suggest solutions.

In this work we introduce **BARBECUE** (a jo**B** **A**wa**R**e **B**ig-data **E**lasticity **C**lo**U**d manag**E**ment system – **BBQ**), a system that allows a Hadoop MapReduce [4] virtual cluster to automatically adjust its size to submitted jobs under a completion deadline. BBQ supports automated cluster deployment, provisioning and on-the-fly, per-job elasticity for the execution of MapReduce programs. BBQ's backbone is a modified version of the Apache Hadoop project [1], an extremely popular platform which is especially designed for Big Data processing. BBQ allows the automatic a) *deployment* and b) *expansion* of a Hadoop cluster on the cloud to improve performance while processing huge amounts of data in order to meet execution time goals set by the user. Additionally, BBQ introduces a reduce phase optimization to improve the use of its elastic properties, by allowing the number of reduce tasks to be dynamically set to achieve better performance.

II. System Architecture

BBQ features an architecture that is illustrated in Fig. 1. It can be used to set up a new virtual Hadoop cluster from scratch and manage its resources taking into account the submitted jobs and their execution deadlines. When a new job is submitted, BBQ's Decision Making module decides on cluster resize operations, if those are required for the execution to successfully meet the deadline. The Decision Making module then interacts with the Cloud Management module which contacts the cloud vendor to adjust the cluster's physical resources by acquiring more virtual machines if necessary. The Cluster Coordinator module executes higher level *add* and *remove node* commands to modify cluster size.

A. Cloud Management

This module is responsible for the interaction with the cloud vendor whenever cluster nodes are to be acquired or released upon request from the Decision Making module. Our cloud management platform is a private OpenStack installation. BBQ can be deployed in Amazon's EC2 or in any EC2-compliant IaaS cloud. To set up new clusters, we have created an Amazon

978-1-5090-6611-7/17 $31.00 © 2017 IEEE
DOI 10.1109/CCGRID.2017.140

Machine Image (AMI) that contains a pre-cooked version of BBQ-flavored Hadoop.

B. Cluster Coordinator

VM coordination is performed with the remote execution of shell scripts and the injection of on-the-fly created Hadoop-specific configuration files to each VM upon creation. Every time the Cloud Management module allocates new VMs, the Cluster Coordinator module is responsible for their integration with the existing nodes of the cluster. Similarly, when nodes are decommissioned, they are first removed from the cluster and then released.

C. Decision Making Module

This module is responsible for deciding the appropriate cluster resize action and number of reduce waves. Its decision depends on the submitted job, input data size and user defined execution time goal. The decision making module is pluggable and can be replaced by a different implementation of BBQ's functionality. The implemented Decision Making module for the purposes of this demonstration uses *hill climbing* to solve the optimization problem of estimating the least nodes necessary to complete a submitted MapReduce job within a given deadline as well as the number of reduce waves of the job which will be executed. To model execution time, we have used the approach thoroughly presented in Section III. *Hill climbing* traverses a space of possible solutions, starting off of the initial state of the cluster, in order to converge to the one which gives us the best approximation of the optimal solution that our model can achieve. Upon reaching a resize decision, the module requests the addition or removal of a number of VMs using the Cloud Management and Cluster Coordinator modules. Once the MapReduce program has been executed and no other job has been submitted for a certain period, the Decision Making Module forces the cluster to gradually shrink by releasing unnecessary idle nodes. Other possible approaches to the problem in hand are presented in works mentioned in Section VI.

D. Hadoop

BBQ uses a modified, cloud aware version of Hadoop. Every time a new job is submitted for execution, Hadoop notifies BBQ, giving it information on the map reduce program to be executed, input data size and maximum desired execution time. Subsequently, the submitted job is paused until the Decision Making module has discovered the optimal cluster configuration and the resizing – if necessary – has been completed. Additional code has been injected in specific classes of the Hadoop framework to communicate with BBQ to notify it of incoming jobs, then wait for the Decision Making module to respond with the number of ReduceTasks to be scheduled and launched and, finally, make the necessary changes before the job is executed.

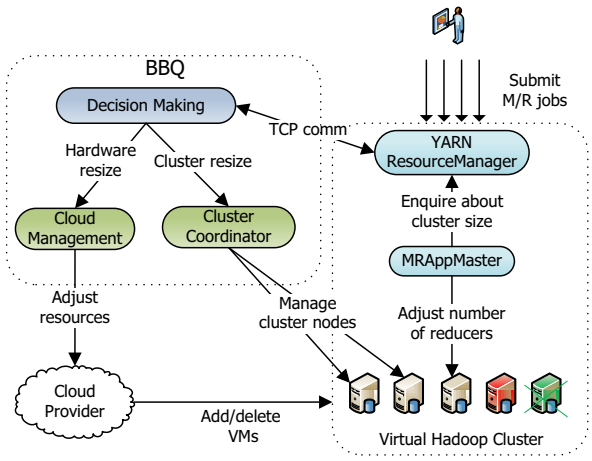


Fig. 1: BBQ architecture

III. Model and Algorithm

A. Performance Model

In order to be able make the required provisioning decisions which will allow Hadoop to complete MapReduce jobs in a specified time window, BBQ's decision making model needs to be able to predict job execution time given a MapReduce program and its input. To accomplish that, we use a Hadoop MapReduce Performance Model, which can express the execution time of MapReduce jobs as a function of those features [16]. To construct our model, we analyze Hadoop's internal functions. To fit the model to specific MapReduce programs, on the other hand, we use performance metrics from past executions – we record the execution time, number of nodes and number of reduce waves the job launched.

There is a set number of YARN `MapTask` and `ReduceTask` `Containers` which can co-exist in a Hadoop cluster at any given point in time. This number depends on cluster size, i.e., the number and size of the cluster's nodes. Let's assume there can be at most m Map `Containers` and r Reduce `Containers` distributed over all cluster nodes. By default, in Hadoop each Map process handles a single block of input data. Therefore, if we have M blocks, M `MapTasks` in total will be scheduled and assigned to m `Containers`. In the ideal case, exactly m `MapTasks` occupy the m `Containers` and run in parallel, resulting in a single *wave* of Map processes. If $M > m$, which is usually the case for large datasets, $\lceil M/m \rceil$ Map waves are needed. In contrast to the total number of Map processes, the number of Reduce processes, R, can be set by the user and is determined by the application requirements. Similarly, if $R > r$, more than one wave of `ReduceTasks` are scheduled. In addition to the cost of the Map and Reduce phases, we should also take into account the tasks of managing and scheduling the M `MapTasks` and R `ReduceTasks`.

1) Map Phase: The Map phase is divided into four sequential operations: *Read*, *Map*, *Sort/Partition*, and optionally *Combine*. Each task's first operation is to *Read* a block of data from the disk – either a local or a remote data block.

We assume the average cost is a function of the size of data block, $b : i(b)$. *Map* is the execution of user defined Map function using the data block read in Read, the complexity of which is determined by the input data size b, denoted as $f(b)$. The Map function output data $o_m(b)$ is often a linear function to the input size b. The output will be a list of $\langle key, value \rangle$ pairs which will be *Sorted* by the key and *Partitioned* into R shares for the R Reduce processes. We denote the cost of partitioning and sorting with $s(o_m(b))$. Since the partitioning process uses a hash function to map the keys, the cost $s(o_m(b))$ is independent of R. The overall cost of a Map process is the sum of the costs mentioned above – without the Combiner component, which will be discussed later:

$$\Phi_m = i(b) + f(b) + s(o_m(b)). \tag{1}$$

This cost is only related to the size of the data block b and the complexity of the Map function. It is independent of the parameters M, R and r.

2) Reduce Phase: The Reduce phase is broken into the following operations: *Copy*, *Merge/Sort*, *Reduce* and *WriteResult*. To simplify our model we will assume the k keys of the Map result are equally distributed to the R Reduce processes. During *Copy*, each `ReduceTask` pulls its shares, i.e., k/R keys and the corresponding records, from the M Map phase outputs. Thus, the total amount of data in each Reduce will be:

$$b_R = M \cdot o_m(b) \cdot k/R. \tag{2}$$

The *Copy* cost is linear to b_R, denoted as $c(b_R)$. A *Merge/Sort* follows to merge the M shares from the Map results while keeping the records sorted, which has the complexity $O(b_R \log b_R)$, denoted as $ms(b_R)$. The *Reduce* function processes the data with some complexity $g(b_R)$ that depends on the application. Assume the output data of the *Reduce* function has an amount $o_r(b_R)$, which is often less than b_R . Finally, the result is duplicated and *Written* back to multiple nodes, with the complexity linear to $o_r(b_R)$, denoted as $wr(o_r(b_R))$. In summary, the cost of the Reduce process is the sum of the component costs,

$$\Phi_r = c(b_R) + ms(b_R) + g(b_R) + wr(o_r(b_R)). \tag{3}$$

Symbol	Meaning
b	single `MapTask` input size
b_R	single `ReduceTask` input size
M	total `MapTasks`
R	total `ReduceTasks`
m	single wave cluster `MapTask` capacity
r	single wave cluster `ReduceTask` capacity
w_r	number of `ReduceTask` waves
N	number of cluster nodes

TABLE I: Symbols used in this section and their meanings.

Algorithm 1 BBQ's Hill Climbing

```
procedure MAKE DECISION(prog, N, M, t_lim, N_max)
    w_r ← 1
    t ← T(prog, N, M, w_r)
    while t > t_lim and N ≤ N_max do
        w_r ← w_r + step_w
        t' ← T(prog, N, M, w_r)
        if t' < t then
            t ← t'
        else
            w_r ← 1
            N ← N + step_N
            t ← T(prog, N, M, w_r)
    return (N, w_r)
```

3) Full Model: Replacing from the equations above, we can see that the overall time complexity T depends on the number of Map waves and Reduce waves. By including the cost of managing and scheduling the Map and Reduce processes $\Theta(M, R)$, which is assumed to be linear to M and R, we represent the overall cost as:

$$T = \lceil \frac{M}{m} \rceil \Phi_m + \lceil \frac{R}{r} \rceil \Phi_r + \Theta(M, R). \tag{4}$$

We want to focus on the relationship that connects the total time cost T, the input data size $M \times b$, the number of `ReduceTasks` R, and the number of YARN `Containers`, m and r. Using a fixed block size b in the analysis, we observe that the cost of each Map process, Φ_m, is fixed. The cost of each Reduce process, Φ_r, is subject to the factor M and R. After replacing and keeping only the variables M, R and m in the model, get:

$$\begin{aligned} T_1(M, m, R, r) = \beta_0 &+ \beta_1 \lceil \frac{M}{m} \rceil + \beta_2 \lceil \frac{R}{r} \rceil \frac{M}{R} \\ &+ \beta_3 \lceil \frac{R}{r} \rceil \frac{M}{R} \log(\frac{M}{R}) + g\frac{M}{R} \\ &+ \beta_4 M + \beta_5 R + \epsilon. \end{aligned} \tag{5}$$

where β_i are the parameters describing the constant factors. $T_1(M, m, R)$ is not linear to its variables, but it is linear to the transformed components: M/m, M/R, $\frac{M}{R}\log(\frac{M}{R})$, $g(M/R)$, M, and R. The parameter β_i defines the contribution of each fine grain operation of either phase in the model. Concretely, β_1 represents the fixed Map cost Φ_m ; β_2 represents the parameter associated with the cost of Copy and WriteBack in the Reduce phase; β_3 represents the parameter associated with the MergeSort component in the Reduce phase; β_4 and β_5 represent the parameters for the cost associated with the management cost $\Theta()$, i.e., we assume the cost is linearly associated with the number of Map and Reduce tasks: $\Theta(M, R) = \beta_4 M + \beta_5 R$; β_0 represents some constant, and ϵ represents the noise component that covers the unknown or unmodeled factors in the system. We discuss the impact of $g(M/R)$ in the model later.

Combiner function. In the Map phase, the Combiner function is used to aggregate the results by key. If there are

k keys in the Map output, the Combiner function reduces the Map output to k records. The cost of the Combiner is only subject to the output of the Map function. Thus, it can be incorporated into the parameter β_1. Additionally, the Combiner function reduces the output data of the Map process and thus affects the cost of the Reduce phase as well. When using a Combiner function, the data that a Reduce process needs to consume from the Map output shrinks to :

$$b_R = M\frac{k}{R}, \tag{6}$$

on average – assuming normal distribution. Since the factor M/R is already present in (5), the cost model applies without changes.

Function $g()$**.** The complexity of the Reduce function has to be estimated each time for the given application. There are some special cases where $g()$ can be removed from (5). If $g()$ is linear to the size of the input data, then its contribution can be merged to the factor β_2, because $g(M/R) \sim M/R$. Similarly, if its complexity is $O(\frac{M}{R}\log(\frac{M}{R}))$, its contribution can be merged to β_3. In those two special cases, the cost model can be simplified to:

$$\begin{aligned} T_2(M,m,R,r) = \beta_0 &+ \beta_1\lceil\frac{M}{m}\rceil + \beta_2\lceil\frac{R}{r}\rceil\frac{M}{R} \\ &+\beta_3\lceil\frac{R}{r}\rceil\frac{M}{R}\log(\frac{M}{R}) + \beta_4 M \\ &+\beta_5 R + \epsilon. \end{aligned} \tag{7}$$

VM creation overhead. The time required for the new VMs to be created, configured and integrated in the cluster is calculated by adding the average time for new VM creation in our cloud infrastructure to the average configuration and integration time for the working cluster. These metrics have been collected through experimentation. Since all VM instances are launched in parallel, we only need to take this term into account once per cluster resize action. According to Mao et al. [12], VM startup time can be considered of trivial cost and should not cause substantial problems. They claim that launching VMs in a cloud environment should not be a bottleneck, as long as the precooked images of their hard drives are not extremely large. In this case these images only include an installation of the operating system and Hadoop. Lagar-Cavilla et al. [10] also made a strong case of VM launch time becoming more and more insignificant.

$$T_{VM} = T_{cloud} + T_{config}. \tag{8}$$

Data Locality. The time required for the new VMs' file system to be populated by the data which will be processed by the Map function is modeled to be linear to the number of new nodes BBQ has added to the cluster. This overhead also depends on the speed of data transfer within the cloud infrastructure and its architecture.

$$T_{init} = \beta_6(N - N_{init}). \tag{9}$$

Training. BBQ uses regression and performance metrics from previous executions of instances of a MapReduce program with various values for (M, R, m, r) to calculate β_i. These model coefficients are stored in a local database and can be updated in regularly for improved fitting.

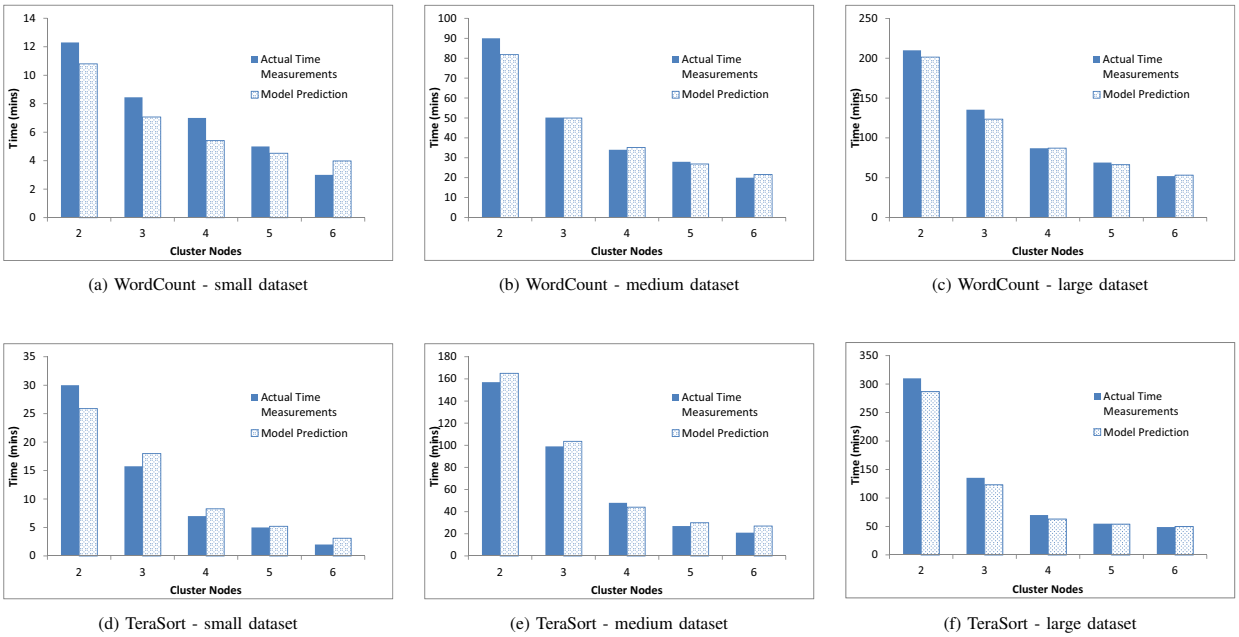
B. Algorithm

Having access to Performance Models for MapReduce programs, our system uses an implementation of *hill climbing* (algo. 1) to solve the optimization problem of minimizing the cluster resources while ensuring job execution time will not exceed its limit. In this case we search in a two dimensional space for pairs of (N, w_r). Starting from the current state, we examine whether increasing the candidate number of reduce waves without adding any nodes can have a positive effect to the performance predicted by our model. As long as it does, we keep increasing the number of reduce waves until the point where that increase has a negative effect in execution time (i.e., the sceduling and networking cost overhead dominates any possible gain) or the model responds with an acceptable execution time. In the case it does not, we increase the candidate number fo cluster nodes and reset the reduce waves to one. We repeat the process until either the model responds with an acceptable execution time or the upper limit for cluster nodes is reached. That limit guarantees that the algorithm will eventually finish. If the algorithm finishes without finding a solution within our cluster's maximum nodes, BBQ deploys the maximum number of nodes allowed. The cost of the algorithm is linear to the number of N values it examined as possible solutions despite the fact that we are in fact searching in a bi-dimensional space, since the number of Reduce waves w_r usually takes a small value over a few iterations – which is expected, since increasing the number of Reduce waves out of proportions can cause a significant overhead in resource managing and data transfer through the web. *Hill climbing*'s overhead is insiginificant in comparison to the processing time for meaningfully large datasets.

IV. Experiments

In this section we will describe our experimental testing of BBQ. In the first part of the our evaluation, we trained our model and tested its accuracy. For that purpose we used two popular Hadoop benchmarks, WordCount and TeraSort[15]. Both these programs' Reduce method is linear to its input, as `MapTasks` read and parse all inputs and send them to reducers. Therefore, we can consider that our simplified model, presented in (7) stands. For the second part of the evaluation, we submitted WordCount and TeraSort MapReduce jobs to a small cluster with a deadline each, observed BBQ in action and recorded its performance.

Experimental Testbed. The experiments were conducted on a private OpenStack cloud. Each worker node had four processing cores and 4GB memory and uses the 2.6.0 version of Apache Hadoop. One node served as the master node and the others as slave nodes. The BBQ deamon ran on a separate dedicated machine. HDFS blocks were set to 256MB.



(a) WordCount - small dataset (b) WordCount - medium dataset (c) WordCount - large dataset

(d) TeraSort - small dataset (e) TeraSort - medium dataset (f) TeraSort - large dataset

Fig. 2: Execution time Prediction for WordCount and TeraSort.

BBQ had been configured taking into consideration specific details of the cluster properties and configuration such as node capacity in terms of memory and cores and resources allocated to YARN `Containers`. `MapTask` and `ReduceTask` `Containers` have the same size and each node can host 3 in total at most. In these experiments for simplicity reasons we used an homogenous cluster but BBQ can also work for heterogenous clusters, since each node is treated as a container for YARN `Containers`. We performed our experiments in small clusters of 2-6 nodes.

A. Model Accuracy

The experimental evaluation of BBQ took place in two distinct steps, the first of wich was the evaluation of the model's accuracy. After calculating the model coefficients for the two benchmarks through a training procedure, we tested the Performance Model by estimating execution times for the following test sets: we used three sets of input data - a small (<30 blocks), a medium size (<100 blocks) and a large one (<200 blocks). We recorded the model's performance and present it in Figure 2. Using R^2 as a measure of goodness of fit [3], we get an acceptable measure of 0,98% for both benchmarks.

B. Evaluating BBQ

We tested BBQ by submitting jobs to a small cluster of two nodes. We measured the execution times of these jobs and compared them to the desired execution times. BBQ's provisioning actions and the actual execution times are presented in Table II.

We notice that in the cases where no cluster expansion is required, most of the times programs complete their execution significantly faster than required. We also observe that a few

Program	BBQ action	Input Size (blocks)	Time goal (mins)	Actual Time (mins)
WordCount	-	27	30	19
WordCount	+1	43	20	18
WordCount	+1	52	20	22
WordCount	+2	71	25	28
TeraSort	-	40	30	22
TeraSort	+1	57	35	32
TeraSort	+3	72	40	41
TeraSort	+4	90	40	44

TABLE II: BBQ resizing the cluster.

of the rest of the executions finish sooner than expected. Since each node can host multiple YARN `Containers`, even adding only one extra node can have a significant impact, even more so in small clusters like the one we experimented with. In most cases our system predicts the execution time quite well, despite missing the requested goal on a few occasions by a short margin. It is more likely for BBQ to miss a given deadline when the cluster resize action it is required to perform significantly changes the cluster size because of the overhead of the initial network data traffic.

V. Demonstration Description

The demonstration will allow attendees to observe BBQ while it is executing MapReduce programs. A simple comprehensive GUI enables users to enter the initialization parameters regarding cluster and VM size. Because the training process is a particularly long one, for the needs of this experiment we will have already completed it, so the BBQ cluster will already

have access to the MapReduce Performance Models for the programs used as benchmarks here, WordCount and TeraSort. In order to launch an experiment on an existing cluster, users will be able to select the program they want to execute, data input size and execution time performance goal.

Attendees can observe how BBQ handles the job submission and resizes the cluster through the GUI provided and also compare the execution time goal with our system's real execution time. They will also be able to access the cluster's Hadoop web interface and monitor the cluster live using performance stats collected and visualized by Ganglia [13].

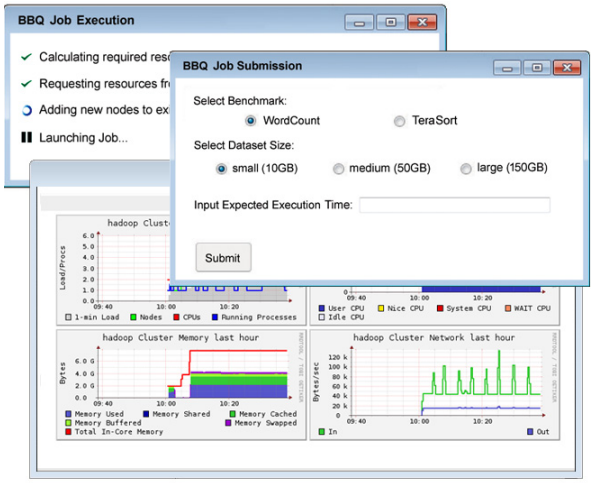


Fig. 3: BBQ UI

VI. Related Work

In recent years there has been a significant production of literature on the topic of *Performance Modeling* in the MapReduce framework. Most authors focus on modifying existing cluster configurations to achieve optimal performance. In [16] Tian and Chen aim to predict performance of a single MapReduce program by using metrics of test runs on a cluster with a smaller number of nodes. They consider MapReduce processing at fine granularity, by partitioning map and reduce tasks into 4 functions each. We have used the idea the authors introduced in this work to form our Performance Model. ARIA [17], introduces a method for Performance Modeling and uses it to implement a deadline-based scheduler for Hadoop. This system creates detailed job profiles from past executions of MapReduce programs and predicts job completion time as a function of allocated resources. The scheduler's goal is to allocate resources to running programs so that they finish within given deadlines, very similarly our system's goal. Khan et al. [8] have recently presented a work where they improve on ARIA and build a system simiar to the one presented in this paper. However their approach is bound to Hadoop-1.*. Our approach leverages the abstraction of YARN, making it feasible for integration with other execution frameworks, such as Spark [18]. The Starfish [6], [7] project applies dynamic instrumentation to collect detailed run-time monitoring data about job execution at a fine granularity, separating jobs into the following eight steps: data reading, map processing, spilling, merging, shuffling, sorting, reduce processing and writing. This data enables the authors to make a thourough analysis and predict job execution under different configuration parameters. Starfish comes with an engine which can suggest the optimal configuration. MRONLINE [11] uses a hill climbing algorithm in order to optimize and on-line fine tune a YARN cluster for the execution of a specific job and is applicable for jobs that even only run once.

Acknowledgment

This paper is supported by European Union's Horizon 2020 RIA programme under GA No 690588, project SELIS.

References

[1] Apache Hadoop. http://hadoop.apache.org/.
[2] IBM-What is Big Data? http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.
[3] A. C. Cameron and F. A. Windmeijer. An r-squared measure of goodness of fit for some common nonlinear regression models. *Journal of Econometrics*, 77(2):329–342, 1997.
[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[5] N. R. Herbst, S. Kounev, and R. Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, 2013.
[6] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *Proceedings of the VLDB Endowment*, 4(11):1111–1122, 2011.
[7] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, volume 11, pages 261–272, 2011.
[8] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang. Hadoop performance modeling for job estimation and resource provisioning. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):441–454, 2016.
[9] I. Konstantinou, E. Angelou, D. Tsoumakos, C. Boumpouka, N. Koziris, and S. Sioutas. Tiramola: elastic nosql provisioning through a cloud management platform. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 725–728. ACM, 2012.
[10] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan. Snowflock: rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 1–12. ACM, 2009.
[11] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, and N. Fuller. Mronline: Mapreduce online performance tuning. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 165–176. ACM, 2014.
[12] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012.
[13] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.
[14] P. Mell and T. Grance. The nist definition of cloud computing. 2011.
[15] O. OMalley. Terabyte sort on apache hadoop. *Yahoo, available online at: http://sortbenchmark. org/Yahoo-Hadoop. pdf,(May)*, pages 1–3, 2008.
[16] F. Tian and K. Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. *2011 IEEE 4th International Conference on Cloud Computing (CLOUD)*, pages 155–162, 2011.
[17] 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*, pages 235–244. ACM, 2011.
[18] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.