ALOJA-ML: A Framework for Automating Characterization and Knowledge Discovery in Hadoop Deployments

Josep Ll. Berral
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain

Nicolas Poggi
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain

David Carrera
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain

Aaron Call
Barcelona Supercomputing Center (BSC)
Barcelona, Spain

Rob Reinauer
Microsoft Corporation
Microsoft Research (MSR)
Redmond, USA

Daron Green
Microsoft Corporation
Microsoft Research (MSR)
Redmond, USA

ABSTRACT

This article presents ALOJA-Machine Learning (ALOJA-ML) an extension to the ALOJA project that uses machine learning techniques to interpret Hadoop benchmark performance data and performance tuning; here we detail the approach, efficacy of the model and initial results. The ALOJA-ML project is the latest phase of a long-term collaboration between BSC and Microsoft, to automate the characterization of cost-effectiveness on Big Data deployments, focusing on Hadoop. Hadoop presents a complex execution environment, where costs and performance depends on a large number of software (SW) configurations and on multiple hardware (HW) deployment choices. Recently the ALOJA project presented an open, vendor-neutral repository, featuring over 16.000 Hadoop executions. These results are accompanied by a test bed and tools to deploy and evaluate the cost-effectiveness of the different hardware configurations, parameter tunings, and Cloud services[1].

Despite early success within ALOJA from expert-guided benchmarking [13], it became clear that a genuinely comprehensive study requires automation of modeling procedures to allow a systematic analysis of large and resource-constrained search spaces. ALOJA-ML provides such an automated system allowing knowledge discovery by modeling Hadoop executions from observed benchmarks across a broad set of configuration parameters. The resulting empirically-derived performance models can be used to forecast execution behavior of various workloads; they allow *a-priori* prediction of the execution times for new configurations and HW choices and they offer a route to model-based anomaly detection. In addition, these models can guide the benchmarking exploration efficiently, by automatically prioritizing candidate future benchmark tests. Insights from ALOJA-ML's models can be used to reduce the operational time on clusters, speed-up the data acquisition and knowledge discovery process, and importantly, reduce running costs. In addition to learning from the methodology presented in this work, the community can benefit in general from ALOJA data-sets, framework, and derived insights to improve the design and deployment of Big Data applications.

[1]ALOJA's repository and sources at http://hadoop.bsc.es/

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.

DOI: http://dx.doi.org/10.1145/2766XXX.XXXXXXX.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance evaluation*; I.2.6 [**Artificial Intelligence**]: Learning—*Induction,Knowledge acquisition*

General Terms

Experimentation; Measurement; Performance

Keywords

Data-Center Management; Modeling and Prediction; Machine Learning; Execution Experiences; Hadoop

1. INTRODUCTION

Hadoop has emerged as the *de-facto* framework for Big Data processing deployment [2][19] and its adoption continues at a compound annual growth rate of 58% [12]. Even with this impressive trend, deploying and running a cost-effective Hadoop cluster is hampered by the extreme complexity of its distributed run-time environment and the large number of potential deployment choices. It transpires that many of the software parameters exposed by both Hadoop and the Java run-time have quite a pronounced impact on job performance[9, 8, 10] and therefore a corresponding effect on the cost of execution [13]. Selecting the most appropriate deployment pattern for a given workload, whether for on-premise servers or as part of a cloud service, involves a complex set of decisions. Any *a-priori* information presented as either heuristic or as a specific performance prediction could greatly improve the decision making process resulting in improved execution times and reduced costs.

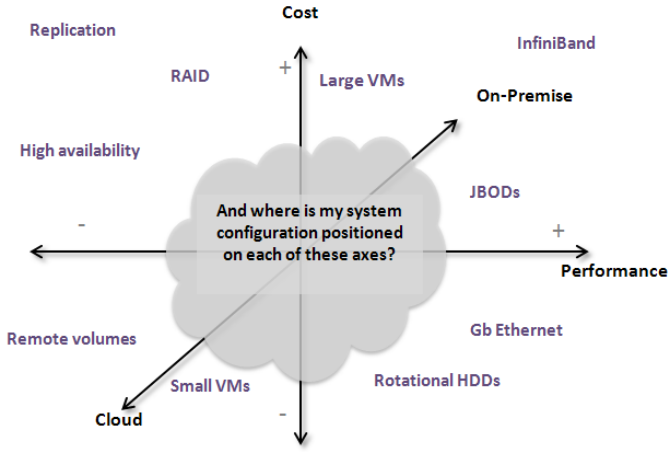

Figure 1: A cloud of points for Cost vs. Performance vs. Cloud or On-premise

Assisting such decision making often requires manual and time-consuming benchmarking followed by operational fine-tuning for which few organizations have either the time or performance profiling expertise. A problem inherent in such complex systems is the difficulty of finding a generalizable *rule of the thumb* for configurations that can be applied to all workloads (Hadoop jobs). To illustrate that complexity, figure 1 presents an example of the search space for evaluating the cost-effectiveness of a particular workload and setup.

ALOJA-ML provides tools to automate both the knowledge discovery process and performance prediction of Hadoop benchmark data. It is the latest phase of the ALOJA initiative which is an on-going collaborative engagement between the Barcelona Supercomputing Center (BSC), Microsoft Product groups and Microsoft Research (MSR). The collaboration explores upcoming hardware architectures and builds automated mechanisms for deploying cost-effective Hadoop clusters. ALOJA-ML's machine-learning derived performance models predict execution times for given a workload and configuration (HW and SW) and provide configuration recommendations for optimal performance of a given task. Given the ability to forecast task performance, the tools can also be used to detect anomalous Hadoop execution behavior. This is achieved using a combination of machine learning modeling and prediction techniques, clustering methods, and model-based outlier detection algorithms.

1.1 Motivation

Dealing with Hadoop optimization necessarily includes running multiple executions and examining large amounts of data, from the information of the environment and configurations, through to the outputs from result logs plus performance readings. In this project we deal with all available data with data mining and machine learning techniques. Selecting which are the relevant features and/or discarding useless information is key to discovering which parameters have the biggest impact on performance, or even the opposite, which features can be freely modified without any detrimental performance impact, allowing the user/operator to adjust configurations not only to me specific performance needs but also to the available resources. Recommending a specific system configuration requires knowledge on how parameter selection will affect various potential goals (e.g., increased performance, reduced execution cost or even energy savings). Modeling and automated learning techniques provide not only the ability to predict the effects of a given configuration, but also they allow exploration of possible configurations *in-vitro*, characterizing how hypothetical scenarios would work. The goal being to demystify the black-art of Hadoop performance tuning by providing Hadoop users and operators with the ability to predict workload performance and empower them with a clear understanding of the effect of configuration selections and/or modifications in both the HW and SW stack.

1.2 Contribution

In ALOJA-ML we aim to provide 1) a useful framework for Hadoop users and researchers characterize and address configuration and performance issues; 2) data-sets of Hadoop experimentation and prediction; and 3) a generalized methodology and examples for automated benchmarking couple to data mining techniques. This is achieved as follows:

1. The ALOJA framework for Big Data benchmarking, including the machine learning enhancements, is available to all the community to be used with own Hadoop data-set executions. Researchers or developers can implement or expand this tool-set through comparing or predicting data from observed task executions and/or by adding new machine learning or new anomaly detection algorithms.

2. All the data-sets collected for ALOJA and ALOJA-ML are public, and can be explored through our framework or used as data-sets in other data analysis platforms.

3. In this work we share our experiences in analyzing Hadoop execution data-sets. We present results on modeling and prediction of execution times for a range of systems and Hadoop configurations. We show the results from our anomalous execution detection mechanisms and model-based methods for ranking configurations depending on relevance. These rankings are used to characterize the minimal set of executions required to model a new infrastructure.

This article is structured as follows: Section 2 presents the preliminaries for this work and current state-of-art. Section 3, presents the datasets, the methodology and results for the Hadoop job execution time predictor model. Section 4 presents two use cases and extension for the predictor model: execution anomaly detection and guided benchmarking, our approach to guide benchmarks executions by predictive value. Finally, sections 5 and 6 the conclusions and future work lines for the project.

2. BACKGROUND

This section presents the background to the project and the current state-of-the art in Hadoop and system performance modeling applying machine learning.

2.1 The ALOJA project

The work presented here is part of the ALOJA Project; an initiative of the Barcelona Supercomputing Center (BSC) that has developed Hadoop-related computing expertise for over 7 years [3]. The efforts are partially supported the

Figure 2: Main components and workflow in the ALOJA framework

Microsoft Corporation, that contributes technically through product teams, financially, and by providing resources as part of its global *Azure4Research* program. ALOJA's initial approach was to create the most comprehensive vendor-neutral open public Hadoop benchmarking repository. It currently features over 16.000 Hadoop benchmark executions, used as the main data-set for this work. The tools on the online platform currently compares not only software configuration parameters, but also *current*, to new available hardware including SSDs, InfiniBand networks, and Cloud services. We also capture the cost of each possible setup along with the run time performance, with a view to offer configuration recommendations for a given workload. As few organizations have the time or performance profiling expertise, we expect our growing repository and analytic tools will benefit Hadoop community to meet their Big Data application needs. The next subsection briefly presets the main platform components, as a more complete description of the scope and goals, the architecture of the underlying framework and the initial findings are presented in [13].

2.1.1 Benchmarking

Due to the large number of configuration options that have an effect on Hadoop's performance, it has been necessary previous to this work to characterize Hadoop using extensive benchmarking. Hadoop's distribution includes jobs that can be used to benchmark its performance, usually referred as *micro benchmarks*, however these type of benchmarks usually have limitation on their representativeness and variety. ALOJA currently features the HiBench open-source benchmark from Intel [11], which can be more realistic and comprehensive than the supplied example jobs in Hadoop. For a complete description please refer to [11], and a characterization of the performance details for the benchmarks can be obtained in the *Performance Charts* section of ALOJA's online application [1].

2.1.2 Current Platform and Tools

The ALOJA platform[2], is composed of open-source components to achieve an automated benchmarking of Big Data deployments, either on-premise or in the cloud. To achieve this goal, clusters and nodes can be easily defined within the framework, and a set of specialized deployment and cluster orchestration scripts take care of the cluster setup. Then, a set of benchmarks to explore and execute can be selected and queued for execution. As benchmarks execute, results are gathered and imported into the main application —that features a Web interface, feeding the repository. On the Web-based repository of benchmark executions, the user can select to search and filter interesting executions and browse the execution details. Advanced analytic features are also provided to perform normalized and aggregated evaluations of up to thousands of results. The tools include:

[2]ALOJA code available at: https://github.com/Aloja/

- Best configuration recommendation for a given SW/HW combination.
- Speed up comparison of SW/HW combinations.
- Configuration parameter evaluation
- Cost/Performance analysis of single and clustered executions
- Cost-effectiveness of cluster setups

Figure 2 presents the main components of the platform that feed each other in a continuous loop. It is the intent of this work to add data-mining capabilities to the current online tools, to enable automated knowledge discovery and characterization, that is presented in later sections.

The platform also includes a *vagrant* virtual machine with a complete sand-box environment and sample executions that is used for development and early experimentation. In our project's site [1] there is further explanation and documentation of the developer tools.

2.2 Related Work

The emergence and the adoption of Hadoop by the industry has led to various attempts at performance tuning an optimization, schemes for data distribution or partition and/or adjustments in HW configurations to increase scalability or reduce running costs. For most of the deployments, execution performance can be improved at least by 3x from the default configuration [8, 9]. A significant challenge remains: to characterize these deployments and performance, looking for the optimal configuration in each case. There is also evidence that Hadoop performs poorly with newer and scale-up hardware [5]. Scaling out in number of servers can usually improve performance, but at increased cost, power and space usage [5]. These situations and available services make a case to reconsider scale-up hardware and new Cloud services from both a research and an industry perspective.

Previous research focused on the need for tuning Hadoop configurations to match specific workload requirements; for example, the Starfish Project from H. Herodotou [10] proposed to observe Hadoop execution behaviors and use profiles to recommend configurations for similar workload types. This approach is a useful reference for ALOJA-ML when modeling Hadoop behaviors from observed executions, in contrast, we have sought to use machine learning methods to characterize the execution behavior across a large corpus of profiling data.

Some approaches on autonomic computing already tackled the idea of using machine learning for modeling system behavior vs. hardware or software configuration e.g., works on self-configuration like J.Wildstrom [20] used machine learning for hardware reconfiguration on large datacenter systems. Similarly, P.Shivam' NIMO framework [16] modeled computational-science applications allowing prediction of their execution time in grid infrastructures. Such efforts are precedents of successful applications of machine learning modeling and prediction in distributed systems workload management. Here we apply such methodologies, not to directly manage the system but rather to allow users, engineers and operators to learn about their workloads in a distributed Hadoop environment.

3. MODELING HADOOP WITH MACHINE LEARNING

The primary contribution of this work is the inclusion of data-mining techniques in the analysis of Hadoop performance data. Modeling Hadoop execution allows prediction of execution output values (e.g., execution time or resource consumption) based on input information such as software and hardware configuration. Modeling the system also enables anomaly detection by comparing actual executions against predicted outputs, flagging as anomalous those tasks whose run-time lies notably outside a machine-learned prediction. Furthermore, compressing system observations using clustering techniques identifies the set of points required for minimal characterization of the system, indicating the most representative tasks needed to model the system or possible sets of tasks needed to learn the behavior of a new system.

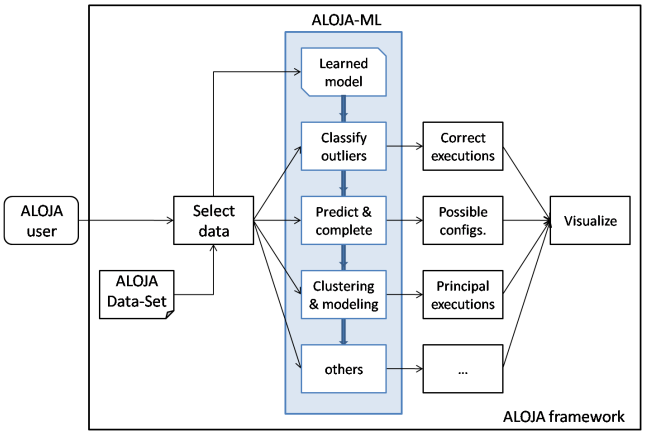

Figure 3: ALOJA-ML inside the ALOJA Framework

The ALOJA framework collates and analyzes data collected from Hadoop task executions and displays it through a range of tools; this helps users understand and interpret the executed tasks. ALOJA-ML complements this framework by adding tools that learn from the data and extract hidden (or not so obvious) information, also adding an intermediate layer of data treatment to complement the other visualization tools. Figure 3 shows the role of ALOJA-ML inside the ALOJA framework.

3.1 Data-sets and Structure

The ALOJA data-set is an open-access collection of Hadoop traces, containing currently over 16.000 Hadoop executions of 8 different benchmarks from the Intel HiBench suite [11]. Each execution is composed of a prepare Hadoop job that generates the data e.g., *teragen*, and a proper benchmark e.g., *terasort*. Although the job of interest is generally the proper benchmark (*terasort*), prepare jobs are also valid jobs that can be also used for training models. This leaves us with over 32.000 executions to learn from. Each benchmark is run with different configurations, including clusters and VMs, networks, storage drives, internal algorithms and other Hadoop parameters. Table 1 shows the data-set relevant features, selected by looking for those providing information into models, and their values.

Figure 4 shows an example execution logged on the ALOJA framework. From here we distinguish the input data, introduced by the user executing the benchmark; the result of execution as the total time spent and time-stamps; and other key information that identifies the execution. We focus our interest on the elapsed time for a given execution as this can determine the cost of execution and indicate whether the execution is successful. Other important output data includes resources consumed such as usage statistics for CPU, memory, bandwidth and storage. We center our focus and efforts on learning and predicting the execution time for a given benchmark and configuration as our initial concern is to reduce the number and duration of executions required to characterize (learn) the system behavior.

Benchmarks	
bayes, terasort, sort, wordcount, kmeans, pagerank, dfsioe_read, dfsioe_write	
Hardware Configurations	
Network	Ethernet, Infiniband
Storage	SSD, HDD, Remote Disks {1-3}
Cluster	# Data nodes, VM description
Software Configurations	
Maps	2 to 32
I/O Sort Factor	1 to 100
I/O File Buffer	1KB to 256KB
Replicas	1 to 3
Block Size	32MB to 256MB
Compression Algs.	None, BZIP2, ZLIB, Snappy
Hadoop Info	Version

Table 1: Configuration parameters on data-set

The collection of traces is part of the ALOJA framework [1] and encourage the academic community and industry to make use of these results and/or to contribute to the corpus of results we have begun to generate.

3.2 Methodology

The learning methodology is a 3-way step model involving training, validation and testing; see Figure 5. Data-sets in ALOJA are split (through random sample) and two subsets used to train and validate the obtained model. A selected algorithm (taken from those detailed in section 3.3) learns and characterizes the system, also identifying and retaining the 'best' parameters (testing them on the validation split) from a list of preselected input parameters. The third subset is used to test the best-from-parameters model. All learning algorithms are compared through the same test subset.

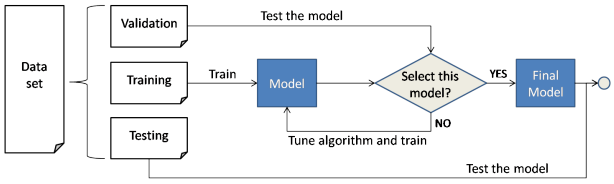

Figure 5: Data-set splitting and learning schema

The tool-set is available as an R library in our on-line *github* repository[3]. In our system the tools are called from the ALOJA web service but access can be initiated from any R-based analysis tool through R's ability to load remote

[3]https://github.com/Aloja/aloja-ml

id_exec	id_cl	bench	exe_time	start_time	end_time	net	disk	bench_type	maps
2	3	terasort	472.000	2014-08-27 13:43:22	2014-08-27 13:51:14	ETH	HDD	HiBench	8

iosf	replicas	iofilebuf	compression	blk_size	# data nodes	VM_cores	VM_ram	validated	version
10	1	65536	None	64	9	10	128	1	1

Figure 4: Example of logged observation on the data-set

functions from the Web. In this way any service or application can call our set of methods for predicting, clustering or treating Hadoop executions. In addition, our machine learning framework can be embedded on Microsoft Azure-ML services, delegating most of the learning process to the Cloud thereby reducing the ALOJA framework code footprint and enabling scaling the cloud. The architecture can be seen in Figure 6. The learning algorithms used here are part of the R *stats* and *nnet* packages, and RWeka [7].

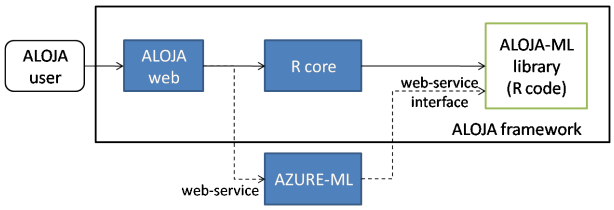

Figure 6: Schema of AZURE-ML on the ALOJA framework

3.3 Algorithms

At this time, the ALOJA-ML user can choose among four learning methods: Regression Trees, Nearest Neighbors, Feed-Forward Artificial Neural Networks, and Polynomial Regression. Each of these has different mechanisms of learning and strengths/weaknesses in terms of handling larges volume of data, being resilient to noise, and dealing with complexity.

Regression tree algorithm: we use the M5P [14, 18] from the RWeka toolkit. The parameter selection (number of instances per branch) is selected automatically after comparing iteratively the prediction error of each model on the validation split.

Nearest neighbor algorithm: we use the IBk [4], also from the RWeka toolkit. The number of neighbors is also chosen the same way as parameters on the regression trees.

Neural networks: we use a 1-hidden-layer FFANN from *nnet* R package [17] with pre-tuned parameters as the complexity of parameter tuning in neural nets require enough error and retrial to not provide a proper usage of the rest of tools of the framework. Improving the usage of neural networks, including the introduction of deep learning techniques, is in the road-map of this project.

Polynomial regression: a baseline method for prediction, from the R core package [15]. Experiences with the current data-sets have shown that linear regression and binomial regression do not produce good results, but trinomial approximates well. Higher degrees have been discarded because of the required computation time, also to prevent over-fitting.

3.4 Execution Time Prediction

Predicting the execution time for a given benchmark and configuration is the first application of the toolkit. Knowing the expected execution time for a set of possible experiments helps decide which new tasks to launch and their priority order (in case of time constraints or the need for immediate insights).

As said previously, the ALOJA data-set is used as training, validation and testing, with separate data splits blindly chosen. Deciding the best sizes for such splits becomes a challenge, as we seek to require as few benchmark executions as possible to train while maintaining good prediction accuracy. Also, at this time, we would like to execute as few jobs as possible to predict the rest of possible executions. Put simply, we look to build an accurate model from the minimum number of observations. As this specific problem is to explore in the next subsection 4.2, here we check the accuracy of predictors given different sizes of training sets. Figure 7 shows the learning and prediction data flow.

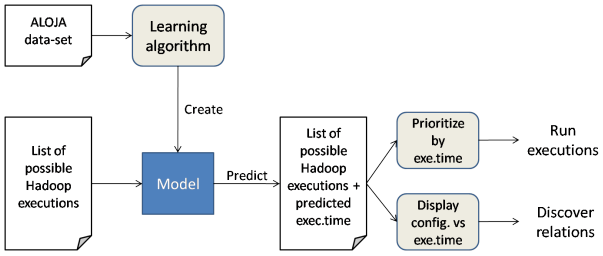

Figure 7: Learning and prediction data-flow schema

Furthermore, in this case we observe how much we can learn from logs and traces obtained from Hadoop executions, how much we can generalize when having different benchmark executions, also how ML improves prediction in front of other *rule of the thumb* techniques applied in the field.

3.4.1 Validation and Comparisons

As explained in subsection 3.3, we have several prediction algorithms from which to create our models, as well as different parameters and choices on the training type. Figure 8 shows the average errors, absolute and relative, for the validation and the testing process for each learning algorithm. In a first exploration, we iterated through some pre-selected parameters and different split sizes for training, validation and testing data-sets. Figure 9 shows how the size of training versus validation affects the model accuracy.

As can been seen in figure 9, using regression trees and nearest neighbor techniques we can model and predict the execution time for our Hadoop traces. We consider that, with a more dedicated tuning, neural networks and deep believe learners could improve results. After testing, linear and polynomial regressions were set aside they achieve poor results when compared with the other algorithms and the time required to generate the model is impractical for the amount of data being analyzed.

Another key concern was whether a learned model could be generalized, using data from all the observed benchmarks,

Algorithm	MAE Valid.	RAE Valid.	MAE Test	RAE Test	Best parameters
Regression Tree	135.19523	0.16615	323.78544	0.18718	M = 5
Nearest Neighbors	169.64048	0.18968	232.01521	0.18478	K = 3
FFA Neural Nets	189.60124	0.24541	333.93250	0.26099	5 neurons (1-hl), 1000 max-it, decay $5 \cdot 10^{-4}$
Polynomial Regression	167.98270	0.2321720	354.93680	0.2541475	degrees = 3

Figure 8: Mean and Relative Absolute Error per method, on best split and parameters found

Algorithm	RAE 50/25/25 Split	RAE 37.5/37.5/25 Split	RAE 20/55/25 Split
Regression Tree	0.18718	0.17903	0.22738
Nearest Neighbors	0.18478	0.21019	0.30529
FFA Neural Nets	0.27431	0.26099	0.27564
Polynomial Regression	0.2541475	0.2602514	0.9005622

Figure 9: RAE per method on test data-set, with different % splits for Training/Validation/Testing

or would each execution/benchmark type require its own specific model. One motivation to create a single general model was to reduce the overall number of executions and to generate powerful understanding of all workloads. Also, there was an expectation that our selected algorithms would be capable of distinguishing the main differences among them (e.g., a regression tree can branch different trees for differently behaving benchmarks). On the other hand, we knew that different benchmarks can behave very differently and generalizing might compromise model accuracy. Figure 10 shows the results for passing each benchmarks test data-set through both a general model and a model created using only its type of observations.

Benchmark	RAE General Model	RAE Specific Model
bayes	0.21257	0.16311
dfsio_read	0.45237	0.18472
dfsio_write	0.26468	0.16861
k-means	0.22700	0.20065
pagerank	0.76518	0.17862
sort	0.42930	0.17535
terasort	0.18190	0.19524
wordcount	0.16282	0.17546

Figure 10: Comparative for each benchmark, predicting them using the general vs. a fitted model with regression trees. The other algorithms show same trends

We are concerned about the importance of not over-fitting models, as we would like to use them for predicting unseen benchmarks similar to the ones already known in the future. Also, the fact that there are some benchmarks with more executions conditions the general model. After seeing the general vs specific results, we are inclined to use benchmark-specific models in the future, but not discarding using a general one when possible.

After the presented set of experiments and derived ones, not presented here for space limitations, we conclude that we can use ML predictors for not only predict execution times of unseen executions, but also for complementing other techniques of our interest, as we present in the following section. Those models provide us more accuracy that techniques used as *rules of thumb* like Least Squares or Linear Regressions (achieving LS with each attribute an average RAE of 2.0256 and Linear Regression a RAE of 0.80451).

3.4.2 Applicability

There are several uses on Hadoop scenarios for such prediction capability. One of them is predicting the performance of known benchmarks on a new computing cluster, as far as we are able to describe this new cluster. Having the hardware configuration of such cluster we can predict our benchmark executions with the desired software configuration. In case of a new benchmark entering the system, we can attempt to check if any of the existing models for specific benchmarks fits the new one. And then treat the new benchmark as the known one, or expand or train a new model for this benchmark.

Another important situation is filling values and configurations, or creating new ⟨configuration, execution time⟩ combinations for a benchmark, in order to observe the expected importance of a given parameter or a given range of values on a parameter, with reference to performance.

Further, an important application of such prediction is to know the size in time of a Hadoop workload, aiming to schedule properly the execution inside a cluster. Being able to find the proper combination of parameters for each workload also their allocation in time and placement becomes an interesting problem. Including predictive capabilities into a job scheduling problem can improve consolidation and deconsolidation processes [6], and so reduce resource consumptions by maintaining any quality of service or job deadline preservation.

Finally, the next section presents another specific uses for the predicting model, with high priority in the road-map of the ALOJA project. These are 1) anomaly detection, by detecting faulty executions through comparing their execution times against the predicted ones; and 2) identification of which executions would best model a given Hadoop scenario or benchmark, by clustering the execution observations, and taking the resulting cluster centers as recommended configurations. Next section focuses on these two cases in detail.

4. SELECTED USE CASES

This section exposes two types of experiences on the use of ALOJA with machine learning tools, and some discoveries obtained over the Hadoop data-set.

4.1 Anomaly Detection

Detecting which executions of our data-sets can be considered valid, flagged for revision or directly discarded, is

the second application of the prediction models. Detecting automatically executions susceptible of being failures, or even executions not modeled properly, can save time to users who must check each execution, or can require less human intervention on setting up *rules of thumb* to decide which executions to discard.

With Hadoop workload behaviors modeled, model-based anomaly detection methods are applied. Considering the learned model as the 'rule that explains the system', any observation that does not fit into the model (this is, the difference between the observed value and the predicted value is bigger than expected), is flagged as anomalous. Here we flag anomalous data-set entries as *warnings* and *outliers*. A *warning* is an observation whose error respect to the model is n standard deviations from the mean. An *outlier* is a mispredicted observation where other similar observations are well predicted. I.e an outlier is a warning that, for all its neighbor observations (those ones that differ in less than h attributes, or with Hamming distance $< h$), more than a half are well predicted by the model. Figure 11 shows the anomaly decision making schema.

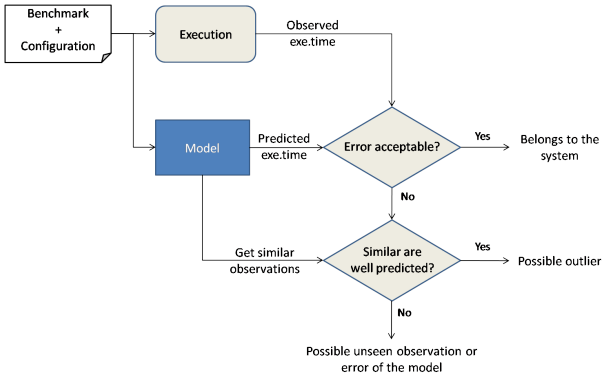

Figure 11: Anomaly detection mechanism schema

In this case we observe how our automatic method can detect outliers or anomalous executions, against a human review and a human-based rules review.

4.1.1 Validation and Comparisons

Once we can model the ALOJA data-set, and knowing that it can contain anomalous executions, we proceed to auto-analyze itself with the anomaly detection method. Here we perform two kind of experiments, one testing the data of a single benchmark with a model learned from all the observations, and one testing it against a model created from its specific kind of observations. We select the observations belonging to the *Terasort* benchmark, with 7844 executions, and the regression trees algorithm from the previous subsection.

After applying our method, using as parameters $h = \{0...3\}$, $n = 3$, and a model learned from all the observed executions, we detect 20 executions of Terasort with a time that does not match with the expected execution time. After reviewing them manually, we detect that those executions are valid executions, meaning that they finished correctly, but something altered their execution, as other repetitions finished correctly and in time. Further, when learning from Terasort observations only (7800), the model is more fitted and is able to detect 4 more executions as anomalous, which in the general model where accepted because of similarities with other benchmark executions with similar times. From here on we recommend to validate outlier executions from models trained exclusively with similar-kind executions.

Testing with different values of Hamming distance shows low variation, as outliers are easily spotted by its error, also confirmed by several neighbors at distance 1. Setting up distance 0, where an observation only is an outlier when there are at least two identical instances with an acceptable time, marks 17:24 observations as warnings, and 7:24 as outliers. Such warnings can be set to revision by a human referee to decide whether are outliers or not. Figure 12 show the comparative of observed versus predicted execution times, marking outliers and warnings.

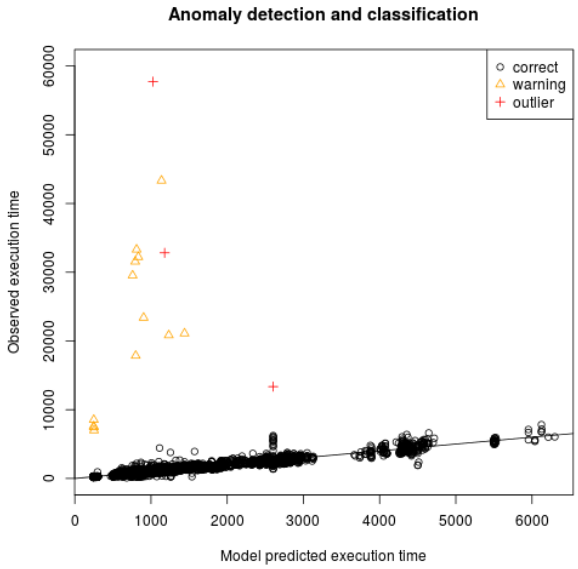

Figure 12: Automatic outlier detection ($h = 0$, $n = 3$)

Notice that the confusion matrix shows anomalies considered legitimate by the automatic method. After human analysis, we discovered that such ones are failed executions with a very low execution time, whose error at prediction is also low, so the method does not detect them. This let us see that for such detection, a manual rule can be set, as if an execution does not exceed a minute, belongs to a failed execution.

Also, we may decide to reduce the number of directly accepted observations by lowering n (standard deviations from the mean) from 3 to 1, putting under examination from a 1% of our data-set up to a 34%. In this situation, we increase slightly the number of detections to 38 (22 warnings and 16 outliers). Figure 13 show the comparative of observed versus predicted execution times, marking outliers and warnings. Also figure 14 shows the confusion matrices for the automatic versus a manual outlier tagging, automatic versus *rule of thumb* (semi-automatic method where ALOJA checks if the benchmark has processed all its data), and automatic warnings versus manual classification as incorrect or "to check" (which is if the operator suspects that the execution has lasted more than 2x the average of its similar executions).

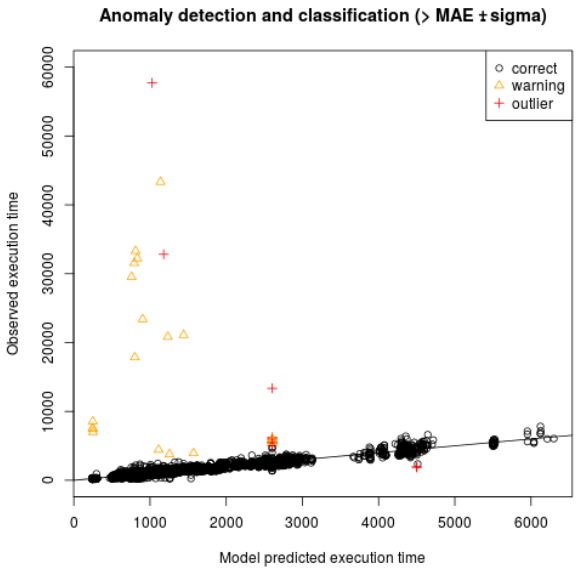

Figure 13: Automatic outlier detection ($h = 0$, $n = 1$)

automatic → manual ↓	Outlier	OK
Anomaly	12	22
Legitimate	4	7786

automatic → *semi-auto.* ↓	Outlier	OK
Anomaly	7	0
Legitimate	9	7786

automatic → manual ↓	Warning	OK
to check	22	0
Legitimate	0	7786

Figure 14: Confusion matrices for different methods

Finally, besides using this method to test new observations, we can re-train our models discarding the observations considered inaccurate. In the first regression tree case, shown in the previous subsection, by subtracting the observations marked as outlier we are able to go from a prediction error of 0.16615 to 0.10873 on validation, and 0.18718 to 0.11912 with the test data-set.

4.1.2 *Use cases*

Spotting failed executions in an automatic way saves time to users, but also let the administrators know when elements of the system are wrong, faulty, or have unexpectedly changed. Further, sets of failed executions with common configuration parameters indicate that it is not a proper configuration for such benchmark; or failing when using specific hardware shows that such hardware should be avoided for those executions.

Furthermore, highlighting anomalous executions is always positive for easing analysis, even more when having more than 16000 executions (and the +300.000 other system performance traces that come with each execution trace). Also it allows to use other learning models less resilient to noise.

4.2 Guided Benchmarking

When modeling a benchmark, a set of configurations, or a new hardware set up, some executions must be performed to observe its new behavior. But executions cost money and time, so we want to run as few sample executions as possible. This is, run the minimum set of executions that define the system with enough accuracy.

We have a full data-set of executions, and we can get a model from it. From it we can attempt to obtain which of those executions are the most suitable to run on a new system or benchmark, and use the results to model it; or we can use the data-set and model to obtain which executions, seen or unseen in our data-set, can be run and used to model. The data-set, as obtained from random or serial executions, can contain similar executions, introduce redundancy or noise. And find which minimal set of executions are the ones that minimize the amount of training data is a combinatorial problem on a big data-set.

Our proposal is to cluster our observed executions (i.e., apply the *k-means* algorithm [15]), obtain for each cluster a representative observation (i.e., its centroid), and use the representatives as the recommended set of executions. Determine the number of clusters (recommendations) required to cover most of the information is the main challenge here. At this time we iterate through a range of k, as figure 15 displays, reconstructing a the model with those recommended observations, and testing it against our data-set or against a reference model. From here on, we decide when the error is low enough or when we exceed the number of desired executions.

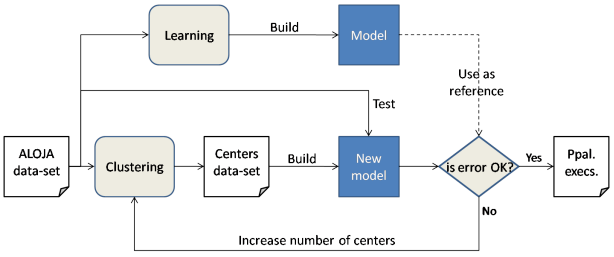

Figure 15: Finding recommended executions schema

4.2.1 *Validation and Comparisons*

For each iteration, looking for k clusters, we compute the error of the resulting model against the reference data-set or model. Also we can estimate the cost of running those executions, as we have the it also we know the average execution cost of 6.85 $\$/hour$ for the clusters we tested. Notice that the estimated execution time is from the seen data-set, and applying those configurations on new clusters or unseen components may increase or decrease the values of such estimations, and it should be treated as a guide more than a strict value. Figure 16 shows the evolution of the error and the execution cost per each k recommendations from our ALOJA data-set. As we expected, more executions implies more accuracy on modeling and predicting, but more cost and execution time.

Further, to test the method against a new cluster addition, we prepared new a setup (on premise, 8 data nodes, 12 core, 64 RAM, 1 disk), and run some of the recommendations obtained from our current ALOJA data-set. We get 6 groups of recommendations with from $k = \{10...60, step = 10\}$, and we executed them in order (first the group of $k = 10$ and so on, removing in this case the repeated just to save experimentation time). We found that with only those 150 recommendations we are able to learn a model with good enough accuracy (tested with all the observed executions of the new cluster), compared to the number of executions needed from the ALOJA data-set to learn with similar accuracy.

Figure 16: Number of recommended executions vs. error in modeling vs. execution cost

Figure 17 shows the comparative of learning a model with n random observations picked from the ALOJA data-set, seeing how introducing new instances to the selected set improves the model accuracy, against picking the new executed instances (this case in order of recommendation groups), and see how it improves the learning rate of the new cluster data-set. We can achieve low prediction errors very quickly, in front of a random execution selection.

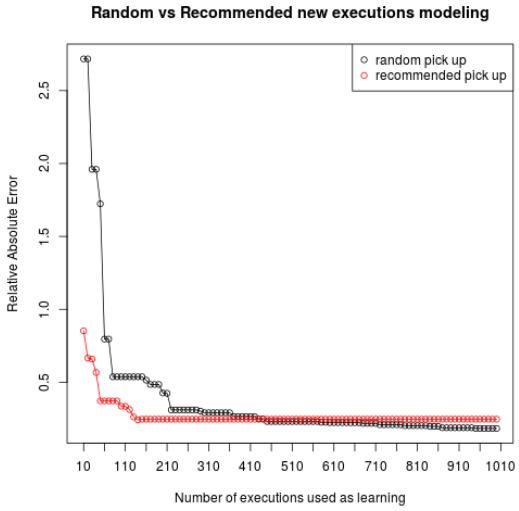

Figure 17: Random vs. recommended executions

4.2.2 *Use cases*

Often executions on a computing cluster are not for free, or the amount of possible configuration (HW and SW) to test are huge. Finding the minimal set of executions to be able to define the behavior of our Hadoop environment helps to save time and/or money. Also, an administrator or architect would like to prioritize executions, running first those that provide more information about the system, and then run the rest in descending order of relevance. This is useful when testing or comparing our environment after modifications, sanity checks, or validating clone deployments.

Further, it is usual when adding new benchmarks or resources, that the new benchmark is similar in behavior to another previously seen, or that a hardware component is similar in behavior to another. Instead of testing it from random executions, we could use the principal executions for the most similar seen environments to test it, and although results can not fit well with previous models (in fact the new environment can be different), use the new observed results as a starting point to create a new model. The study of how it performs well against other example selection methods for Hadoop platforms and brand new benchmarks is in the ALOJA-ML road-map for near future research.

5. SUMMARY AND CONCLUSIONS

In this article we described ALOJA-ML, a rich tool-set for carrying on automated modeling and prediction tasks over benchmarking data repositories, with particular focus on the BigData domain. The goal of ALOJA-ML is to assist performance engineers, infrastructure designers, data scientists writing BigData applications and cloud providers in minimizing the time required to gain knowledge about deployment variables for BigData workloads. ALOJA-ML takes large performance data repositories available in the ALOJA project and through several Machine Learning techniques identifies key performance properties of the workloads. This information can be later leveraged to predict performance properties for a workload execution on a given set of deployment parameters that have not been explored before in the testing infrastructure. In particular, in this paper we take the case of the Hadoop ecosystem, which is the most widely adopted BigData processing framework currently available, to describe the approach used in ALOJA-ML.

The article describes: a) the methodology followed to process and model the performance date; b) the group of learning algorithms selected for the experiments; c) and the characteristics of the public data set (consisting of more than 16,000 detailed execution logs at the time of the experiments

but continuously growing) used for training and validation if the learning algorithms. Through our experiments, we exposed and demonstrated that using our techniques we are able to model and predict Hadoop execution times for given configurations, with a small relative error around 0.20 depending on the executed benchmark. Further we passed out data-set through an automated anomaly detection method, based on our obtained models, with high accuracy respect a manual revision. Also we deployed a new Hadoop cluster, running the recommended executions from our method, and tested the capability of characterizing it with little executions; finding that we can model the new deployment with fewer executions than by randomly selecting executions.

The work presented includes two selected use cases of the ALOJA-ML tool-set in the scope of the ALOJA framework. The fist one is the use of the learning tools for guiding the costly task running experiments for the online performance data-set. In this case, ALOJA-ML assists the framework at the time of selecting the most representative runs of an application that wants to be characterized in a given set of deployment, with the consequent reduction in terms of time and costs to produce relevant executions. The second use case assists in identifying anomalies on large sets of performance testing experiments, which is a common problem of benchmarking frameworks, and that needs to be addressed to get unbiased conclusions on any performance evaluation experiment.

6. FUTURE WORK

The current road-map of ALOJA-ML includes to add new features, tools and techniques, improving the ALOJA framework for Hadoop data analysis and knowledge discovery. Learn how to deal with big amounts of data from Hadoop executions is among our principal interests. This knowledge and capabilities can be used to improve the comprehension and management of such platforms. We are open to new techniques or new kind of applications.

Our next steps are: 1) study techniques to characterize computation clusters and benchmarks, to prepare Hadoop deployments to deal with seen and unseen workloads; 2) introduce new input and output variables, looking new sources of information from the system, and predicting other performance indicators like resource usage; 3) study in detail compatibilities and relations among configuration and hardware attributes, in order to detect impossible setups or executions incompatible with a given deployment; 4) improve the methods to select features, examples and learning parameters. Also introduce new learning algorithms and methodologies like deep-belief networks; 5) and add new executions to the ALOJA data-sets, with new deployments and benchmarks.

7. ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 639595). This work is partially supported by the Ministry of Economy of Spain under contracts TIN2012-34557 and 2014SGR1051.

8. REFERENCES

[1] ALOJA Project. `http://hadoop.bsc.es` (Feb 2015).

[2] Apache Hadoop. `http://hadoop.apache.org` (Jan 2015).

[3] BSC-Autonomic. Autonomic systems and big data research group page. `http://www.bsc.es/computer-sciences/performance-tools` (Jan 2015).

[4] D. Aha and D. Kibler. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[5] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron. Scale-up vs scale-out for hadoop: Time to rethink? In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 20:1–20:13, New York, NY, USA, 2013. ACM.

[6] J. L. Berral, R. Gavaldà, and J. Torres. Power-aware multi-data center management using machine learning. In *42nd International Conference on Parallel Processing, ICPP 2013, Lyon, France*, 2013.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[8] D. Heger. Hadoop Performance Tuning. `https://hadoop-toolkit.googlecode.com/files/White%20paper-HadoopPerformanceTuning.pdf` (Jan 2015).

[9] D. Heger. Hadoop Performance Tuning - A pragmatic & iterative approach. DH Technologies, 2013.

[10] 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*, 2011.

[11] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. *Data Engineering Workshops, 22nd Intl. Conf. on*, 0:41–51, 2010.

[12] L. Person. Global Hadoop Market. Allied market research, March 2014.

[13] N. Poggi, D. Carrera, A. Call, S. Mendoza, Y. Becerra, J. Torres, E. Ayguadé, F. Gagliardi, J. Labarta, R. Reinauer, N. Vujic, D. Green, and J. Blakeley. ALOJA: A systematic study of hadoop deployment variables to enable automated characterization of cost-effectiveness. In *IEEE Intl. Conf. on Big Data 2014, Washington DC, USA*, 2014.

[14] R. J. Quinlan. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, Singapore, 1992. World Scientific.

[15] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.

[16] P. Shivam, S. Babu, and J. Chase. Active and accelerated learning of cost models for optimizing scientific applications. In *International Conference on Very Large Data Bases*, 2006.

[17] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0.

[18] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *9th European Conference on Machine Learning*. Springer, 1997.

[19] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[20] J. Wildstrom, P. Stone, E. Witchel, and M. Dahlin. Machine learning for on-line hardware reconfiguration. In *20th Intl. Joint Conference on Artifical Intelligence*, IJCAI'07, 2007.