From Performance Profiling to Predictive Analytics while Evaluating Hadoop Cost-Efficiency in ALOJA

Nicolas Poggi, Josep Ll. Berral, David Carrera,
Aaron Call, Fabrizio Gagliardi
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catlalunya (BarcelonaTech)
Barcelona, Spain

Rob Reinauer, Nikola Vujic[1],
Daron Green and José Blakeley
Microsoft Corporation, Microsoft Research (MSR)
Redmond, USA
Microsoft Development Center Serbia (MDCS)[1]
Belgrade, Serbia

***Abstract*—During the past years the exponential growth of data, its generation speed, and its expected consumption rate presents one of the most important challenges in IT both for industry and research. For these reasons, the ALOJA research project was created by BSC and Microsoft as an open initiative to increase cost-efficiency and the general understanding of Big Data systems via automation and learning. The development of the project over its first year, has resulted in a open source benchmarking platform used to produce the largest public repository of Big Data results[1], featuring over 42,000 job execution details. ALOJA also includes web-based analytic tools to evaluate and gather insights about cost-performance of benchmarked systems. The tools offer means to extract knowledge that can lead to optimize configuration and deployment options in the Cloud *i.e.*, selecting the most cost-effective VMs and cluster sizes.**

This article describes the evolution of the project focus and research lines, for a period of over a year while continuously benchmarking systems for Big Data. As well discusses the motivation —both technical and market-based— of such changes. It also presents the main results from the evaluation of different OS and Hadoop configurations, covering over 100 hardware deployments. During this time, ALOJA's initial target has shifted from a previous low-level profiling of Hadoop runtime with HPC tools, passing through extensive benchmarking and evaluation of a large body of results via aggregation, to currently leveraging Predictive Analytics (PA) techniques. The ongoing efforts in PA show promising results to automatically model the behavior of systems *i.e.*, predicting job execution times with high accuracy or to reduce the number of benchmark runs needed. As well as for Knowledge Discovery (KD) to find relations among software and hardware components. Techniques that jointly support foresighting cost-effectiveness of new defined systems, reducing benchmarking time and costs.

I. Introduction

The ALOJA project [19] is an open initiative from the Barcelona Supercomputing Center (BSC) to explore and automate the characterization of cost-effectiveness for Big Data deployments. BSC is a center with over 8 years of research expertize in Hadoop environments. The project counts with support from both research Big Data product groups within Microsoft, as well as cloud resources from the *Azure4Reseach* [10] program, and recently from Rackspace Inc [22] and the Future SOC Lab [13].

[1]ALOJA's web application, benchmark repository, tools, and sources available at http://hadoop.bsc.es

ALOJA attempts to provide solutions to an every time more important problem for the Big Data community, which is the lack of understanding of what parameters, either software (SW) or hardware (HW), determine the performance of Big Data workloads. Therefore the selected configuration determines the speed in which data is processed and returned, and most importantly, the hosting budget.

Additionally, as open-source frameworks *i.e.*, Hadoop and Spark become more common, they can be found in a diversity of operational environments. Comprising from low-end commodity clusters, low-powered microservers, to high-end data appliances, including all types of Cloud-based solutions at scale *i.e.*, IaaS and PaaS. Where due to the large number of software configuration options *i.e.*, concurrency and memory buffers [14–16], and the increasing number of deployment types *e.g.*, Azure has over 32 VM types that can be used for Hadoop [8], optimizing the performance of Big Data systems requires extensive manual benchmarking [15, 18]. On top of this large number of SW and HW combinations, Big Data jobs *i.e.*, sorting TBs of data (*terasort*), where each iteration can take hours to complete. Creating a broad search space of configurations with high resource requirements to explore. For these reasons, and the lack of benchmarking automation tools, in ALOJA we have built progressively an open-sourced benchmarking platform briefly presented in Figure 1. Which allows Big Data researchers and practitioners to evaluate the latest HW components, as well as to quickly estimate cost-effectiveness of new market products.

This article presents the evolution of the project, results, and lessons learned while continuously benchmarking Hadoop for over a year. Iterating over software configuration options and covering over a hundred different hardware cluster setups including different server nodes and VMs; cluster sizes; disks arrays, both local and network attached; and networks deployments including InfiniBand. At the same time, it discusses the motivation both technical and market-based of the current project focus i.e., the increasingly large search space and the emergence of economic Cloud services. As well as an overview

Fig. 1. Main components and workflow in the ALOJA platform

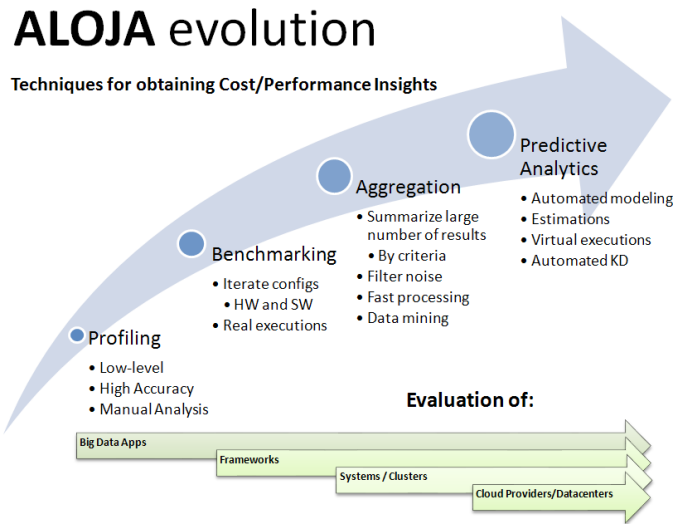

Fig. 2. Evolution of ALOJA: from performance profiling to PA benchmarking

on the different techniques employed to extract performance knowledge and insights from Hadoop executions, as well as it exposes our current lines of research and focus. An overview of the project's evolution summarized in Figure 2, which shows the different performance extraction techniques employed in the project. As well as the expansion to extract knowledge from Big Data applications to infrastructure providers.

Since project beginnings almost two years ago, ALOJA's target has evolved from initially producing low-level profiling traces of Hadoop distributed environment —which allows to understand the networking bottlenecks and how components interact—, to performing extensive benchmarking. The current effort is still expanding, and by now has already produced the largest existing public Hadoop performance repository with over 42,000 job with their performance traces. The repository can be used to study the performance and cost effectiveness of the different configurations available, and several advanced *aggregation* and *summarization* mechanisms are in place to reduce both the size of data to be processed and stored. The platform is shared with other researchers online, or in the form of a development virtual image that can be downloaded and instantiated get a local replica of the ALOJA framework.

While the results from the data *aggregation* efforts allows to process data interactively for the analytic online tools [6], the increasing number of configuration choices as the project expands in architectures and services —in the *millions* for benchmarks that a single iteration can take *hours* to execute— had led us to leverage Predictive Analytics (PA) techniques to be able to prioritize benchmark configurations and reduce the number of executions. PA encompasses a variety of statistical and Machine Learning (ML) techniques to make predictions of unknown events based on historical data [11, 28] —in this case the aggregated metadata of our benchmarking repository. Currently the ongoing PA efforts show promising results to automatically model system's behavior from existing benchmarks *i.e.*, predicting job execution times with high accuracy or to reduce the number of benchmark runs needed. As well as for Knowledge Discovery (KD) to find relations among software and hardware components and provide recommendations for users *e.g.*, the most cost-effective VM type and cluster size in a Cloud provider or the expected speedup of a new disk configuration. Techniques that jointly support foresighting cost-effectiveness of new defined systems, and importantly for the project, reducing benchmarking time and costs by using the prediction models.

II. Online Repository and Platform Tools

Part of initial approach of the project included the creation of vendor-neutral open-public Hadoop benchmark repository. This effort currently features more than 42,000 Big Data job benchmark executions, along with their performance traces and logs. As few organizations have the time or performance profiling expertise, we expect the repository will benefit the Big Data community to meet their application needs without the need of benchmarking. As well as bringing more transparency to product claims and reports, as results are fully disclosed and experiments can be easily repeated, especially for public cloud results. This growing repository is used as the data-set for experimentation, in which we apply traditional modeling and characterization techniques as well as Machine Learning techniques to automate Knowledge Discovery. Most of the tools we develop are also available in the web site, to allow easy sharing of results, that are added as new benchmarks enters the system.

The ALOJA platform is further composed of open-source tools to achieve an automated benchmarking of Big Data deployments. These tools are used by researchers testing new features and algorithms, or by practitioners needing either to privately test their own system and application, or to improve benchmark results. The 3 main components are: Big Data benchmarking scripts (code name *datathlon*), that deploy servers and execute benchmarks; the online repository; and Web Analytics tools, that feed each other in a continuous loop as benchmark are executed; and the list of new runs comes from the PA analytic components as shown in Figure 1. To achieve automation, clusters and nodes can be easily defined within the platform, 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 web application. On the web-based repository of benchmark executions, the user can select to search and filter relevant executions and browse the execution details (if needed). Advanced analytic features are also provided to perform normalized and aggregated evaluations of up to thousands of results. Figure 3 presents the end-to-end process that ALOJA follows to turn benchmarks into knowledge. The tools include:

- Best configuration recommendation for a given SW/HW combination.
- Speed up comparison of SW/HW combinations.
- Configuration parameter evaluation.
- Cluster scalability comparison in relation to cost and performance.
- Cost/Performance analysis of single and clustered executions.

- Cost-effectiveness of cluster setups.

The platform also includes a *vagrant* virtual machine setup with a complete sandbox 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. In this study, we also describe the new data-mining capabilities to the current online tools, which enable automated knowledge discovery and characterization.

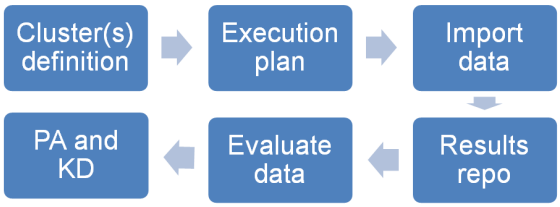

Fig. 3. End-to-end workflow from definition to Knowledge Discovery

III. ALOJA Project Evolution

During the development of the defined phases for ALOJA presented in [19], we have experienced a shift from an initial approach of using low-level High Performance Computing (HPC) tools to profile Hadoop runtime [7], based BSC's previous expertise to higher-level performance analysis. Part of the initial work included inserting hooks into the Hadoop source code to capture application events, that are later post-processed into the format of BSC's HPC tools, which are used to analyze the performance and parallel efficiency of *supercomputing* or MPI-based workloads. However, due to the nondeterministic nature of Hadoop distributed execution, on top of the large number of software configuration options, some of the HPC tools were not directly applicable for finding best configurations for a particular hardware. Furthermore, our target has not been to directly improve the Hadoop framework codebase, but its deployment scenario.

To compare configuration options, during the first months of the project we started an extensive benchmarking effort on different cluster architectures and cloud services; iterating software configurations to extract execution times that were comparable. The resulting repository of benchmarks can be found at the project's web site [6]. As the number of benchmarks in the repository grew, evaluating internal results for each was no longer feasible, either for the low-level performance analysis tools or for manually revising them. Performance metric collection and log parsing both for profiling and benchmarking, have become Big Data problem in itself as results grow. For this reason, aggregation into summaries of the execution characteristics i.e., sums and averages. In this way turning execution details into meta-data, which allow us to contrast different results more efficiently; at the loss of information on execution internals, but reducing processing time. To explore these results more efficiently, we then developed different Web views and filters of the aggregated executions from the repository's online database. The analysis of results has led to a shift of focus in the project from the initial low-level profiling and Hadoop configuration, to cluster configuration and the cost/performance efficiency of the different systems as is shown again in Figure 2.

Another reason for this change in perspective has been a shift from benchmarking on on-premise to Cloud based clusters. Current cloud offerings for Big Data provide compelling economic reasons to migrate data processing to the Cloud [25] with the *pay-as-you-go* or even *pay-what-you-process* models. While the cloud has many benefits for cluster management *i.e.*, dynamically scaling in servers, it also introduces challenges for moving data e.g., data is no longer local to nodes and it is accessed over the network. Furthermore, on the Cloud, due to the virtualized and public *multi-tenant* nature, low-level profiling becomes less relevant, as many samples of the same execution are needed to estimate average performance and results aggregation come into play. On top of this, for Platform-as-a-Service (PaaS) offerings, you might not have superuser access to the server to install packages for profiling the system [2].

Cloud providers also offer a great number of virtual machines (VM) options —at time of writing the Microsoft Azure Cloud offers over 32 different VM choices [8]. Under this model, the same number of CPU processing cores and memory can be achieved by either having a larger number of small VMs (scale-out) or having a few larger VMs (scale-up) in a cluster. This great number of cluster configuration choices, which have an impact in the performance and the costs of executions [18, 29] has become one of the main targets of our research and benchmarking efforts. Examples of cloud VM size comparisons can found in our Web site [6]. These large number of Cloud topologies and services, hardware technologies, software configuration options that affect the execution time —and costs— of the different applications, leave us with millions of possible benchmark executions in the search space.

In order to cope with the increasing number of configuration options and new systems, the project was faced with the need first to do manual sampling from the search space, and grouping of results to extrapolate results between clusters. However, this initial approach has not been sufficient either, and still requires a large number of benchmarks. For this reason, we have leveraged Machine Learning (ML) techniques and implemented them as an extension of the platform. The generated prediction models allow us to estimate metrics such as job execution time for not-benchmarked configurations. Achieving a great accuracy, compared to classical statistical sampling and interpolation, as well as saving us time and execution costs. We are currently in the process of extending the use of such models on the platform to enable the Predictive Analytics (PA) [11, 28] to complement the descriptive analytical tools available. Currently, PA techniques also complement our goal of automating Knowledge Discovery (KD) from the growing benchmark repository.

Having our efforts focused in PA, does not mean that we have stopped benchmarking or low-level profiling efforts. Each of the techniques have different uses cases and can complement each other *i.e.*, PA is based on benchmarking meta-data and profiling is used to debug or improve OS settings and configuration, and to fine-tune components with greater precision. However, PA results allow us to follow market changes and find cost/performance tendencies more rapidly. The next section compares each of the different performance extraction methods identified above.

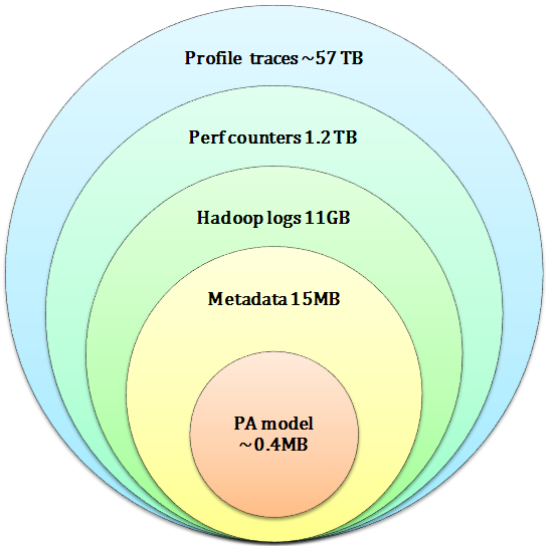

Fig. 4. Comparing the data sizes produced by each strategy (chart not in scale)

IV. Approaches to extract performance knowledge from Hadoop

This section describes distinctive techniques to measure efficiency, performance, resource usage and costs in ALOJA and presents some results and use cases of their application.

A. Profiling

BSC having strong background with HPC workloads and their performance [7], the preliminary approaches to ALOJA consisted in instrumenting Hadoop, to make it compatible with well established performance tools used for parallel supercomputing jobs e.g., for MPI and related programming models. This initial work was achieved by developing the now *Hadoop Analysis Toolkit* which leverages the *Java Instrumentation Suite* (JIS) [12], a tracing environment for Java application, inserting hooks into the Hadoop source code; and a network sniffer based in *libpcap*. These changes allowed us to leverage the already existing low-level tools, simulators and know-how from the HPC world, at the cost of having to patch and recompile Hadoop code. To overcome this limitation, we are currently working implementing dynamic code interposition *e.g.*, Aspect Oriented Programming is currently being worked on. Network traces allow us to study in great detail e.g, up to packet-level, the communication between components; both between nodes, as well as local data transfers as Hadoop uses remote and local services to transfer data.

Employing the *Hadoop Analysis Toolkit* for profiling, we are able to understand at specifically what timeframe certain Map/Reduce (M/R) steps i.e., shuffle, sort, merge, and spill. As well as how components synchronized in the cluster during the different distributed tasks. While profiling allow us a deep understanding of the underlying execution and debugging code, the level of detail can be daunting if not with the intent of optimizing the Hadoop code, or working with drivers or accelerators. Another problem that we faced with profiling, is the large amount of data that it produces. While the overhead to produce it is not a main concern, it leaves us with a Big Data problem in itself. Figure 4 compares the amount of data that would have been produced by profiling if enabled for the 42,000 executions in the repository comparing it to the rest of the techniques employed. Due to the large amount of traces profiling generates, we currently enable low-level profiling only on selected traces, when a particular execution requires a deep level of analysis and debugging.

B. Benchmarking

Due to the large number of configuration options that have an effect on Hadoop's performance, to improve the efficiency of Hadoop requires manual, iterative and time consuming benchmarking and fine tuning over a myriad of software configuration options. Hadoop exposes over 100, interrelated configuration parameters that affect its performance, reliability and operation [15, 18]. These reasons, as well as the lack of automation tools for Big Data had led us to build progressively, an automated benchmarking platform to deal with defining cluster setups, server deployment, defining benchmarking execution plans, orchestration of configuration, and data management of results. While the platform is generic for benchmarking, our use case has been Hadoop v1 and v2.

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 [17], which can be more realistic and comprehensive than the supplied example jobs in Hadoop. HiBench features several ready to use benchmarks from 4 categories: micro benchmarks, Web search, Machine Learning, HDFS benchmarks. We have currently gathered over 42,000 executions from the different HiBench benchmarks, that we use as base of our research to automate characterization and KD for Big Data environments.

In contrast to profiling, the benchmarking efforts operates more as black-box, where different Big Data frameworks and applications can be tested and summaries about their execution time and performance metrics are collected. However, we do include in ALOJA specific features for Hadoop, such as log parsers to detect the different phases in the M/R process, which could be extended for other frameworks if needed. Over time, collecting performance metrics have also become a Big Data problem: we have over 2TB of performance metrics for the executions after importing the executions into the database. A description of the architecture in ALOJA can be found in [19]. While the data is $\sim$ 45x smaller than traces from profiling as can be seen again in Figure 4, as we get more executions, we currently use this data mostly for debugging executions manually. While we still keep it for every execution, summarizing data via aggregation has become more useful to extract cost and performance knowledge from groups of results in our use cases.

1) Benchmark statistics: At this time we have available 8 different benchmarks with their respective preparation processes (*preps*). From this 42.000 executions about half of them are *preps*, and from the benchmark tasks around 2.000 are failed executions, as their execution time did not exceed 100 seconds, or the Hadoop counters are missing (marking the end of correct runs). Table I shows the basic statistics for our

execution database without preps and failed executions. The executions in our database at time of writing includes about 10,000 runs in different *on-premise* clusters. On the cloud as IaaS, it includes more than 25,000 on Microsoft Azure and 4,100 Rackspace. For PaaS cloud, over 500 runs from the HDInsight service in Azure.

	Average	St.Dev	Min.	Max.	#Execs
bayes	844	355	324	1235	331
dfsioe_r	4689	6667	292	96949	1375
dfsioe_w	2062	5960	292	160793	1376
kmeans	2633	11051	442	350829	1250
pagerank	2178	3430	493	88786	1614
sort	2397	4770	497	89306	2022
terasort	1865	8763	207	661167	11340
wordcount	1594	2973	340	86930	2189

TABLE I. EXECUTION TIME PER BENCHMARK (IN SECONDS), AND NUMBER OF EXECUTIONS PER BENCHMARK

When considering *preps* also as tasks to be run, those match their respective benchmark in number of executions, with an average execution time of 360 seconds. In some occasions for *terasort* benchmark, some *preps* are also failed executions. Where the time for preparation exceeds the 1,000 seconds and sometimes reaching up to 10,000 seconds, making the regular execution to fail too. The repository also includes runs where for some reason the execution lasted much longer than the expected, but finishing successfully in most cases (as noticed in the *terasort* benchmark execution time standard deviation). In order to examine the causes of these slow runs, the performance counters and Hadoop logs can be revised on the Web application to pinpoint the causes manually. A common error is task timeouts, usually caused by bottlenecks in the I/O subsystem for some configurations or node failures.

C. Aggregation and Summaries

After a benchmark is executed and its performance traces and logs stored, the produced folder is then imported into a relational database. This process involves decompressing data, transforming formats by using command line tools, parsing Hadoop logs, and importing the data into the database. The import operation is a one time process; however the execution folders are kept in case they need to be re-imported in the future due possible changes to the import routines or the use of external tools compatible with the logs. Once the data is on the relational database, the user can interact with the information using the website component (ALOJA-WEB), connecting to the DB through standard SQL. Through data *aggregation*, the user can visualize and focus on specific details of the collected executions.

As Hadoop job executions depend on the execution context (nondeterministic), and several reasons can affect a particular execution performance i.e., multi-tenancy in the Cloud, or different random data generated by the benchmarks; all repeating executions are grouped and averaged (or other statistical functions are performed such as means or percentiles). Visualization tools in ALOJA-WEB, allow to group together executions that share some common parameters *i.e.*, same number of data-nodes, disk configuration, or number of mappers; to complement missing executions and produce a common result.

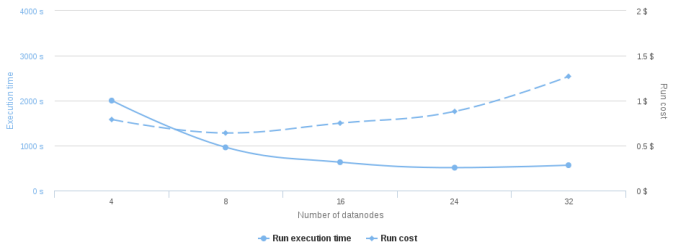

Fig. 5. Scalability of cluster size for terasort

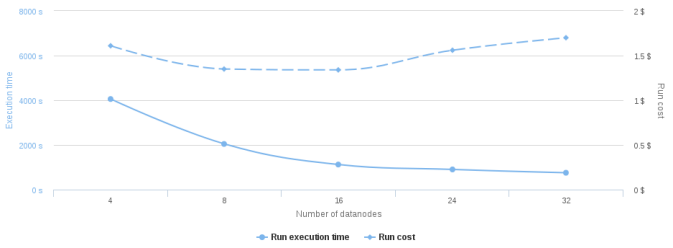

Fig. 6. Scalability of cluster size for wordcount

While these results need to be revised and validated, i.e., mixing executions with different cluster sizes might produce incoherent results; they can quickly offer a view on the tendency for the cost-performance. Another example of the performance evaluations using this technique, Figures 5 and 6 show the scalability of different VMs in the HDI PaaS service from Azure with the *terasort* and *wordcount* benchmarks respectively, by aggregating similar executions and separating them by number of dedicated data-nodes. Visualizations focus in the execution time (see continuous lines and Y-left axis on figures), also on the execution cost (see dotted lines and the Y-right axis). It is seen that while increasing the number of nodes reduces the execution time but after 16 nodes for *wordcount* —and only 8 nodes for terasort— this reduction is lower and costs grow again.

These tools can be found in the *Config. Evaluations* and the *Cost/Perf. Evaluations* menus; including other tools that allow to find the best configurations for particular selections of executions, explore scalability on specific configuration parameters, and rank cluster deployments by cost-performance.

By aggregating results, instead of running all the possible configurations for a benchmark, we can run a sample of them and extrapolate the results to the missing executions, by aggregating them and using averages. Another feature is that it allows us to *average* noise or outlier executions automatically. While this is an improvement to analyzing benchmark result individually, it has several drawbacks: it exposes us to the problems of averages, that might not represent a well distributed value. Also might lead to wrong conclusions if the grouped data is not analyzed and validated manually carefully, as mixing technologies and cluster sizes in these groups. For these reasons, and continue our goal towards automation, we have started the PA extension to overcome some of these shortcomings and automate Knowledge Discovery.

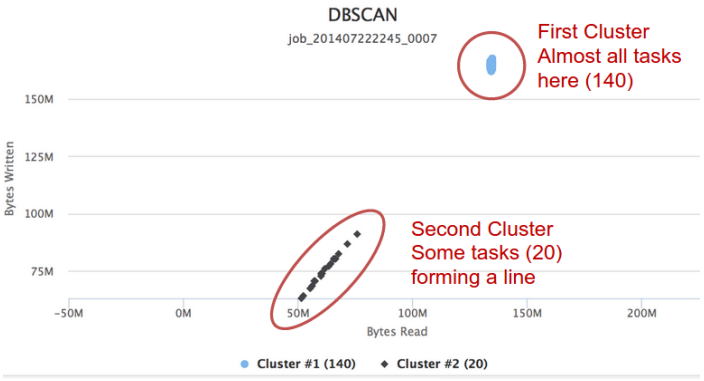

Fig. 7. Example of two different task behaviors on k-means benchmark detected by DBSCAN

D. Initial Data Mining

Our first approach in mining the ALOJA data-set consisted on the *Density-based spatial clustering of applications with noise* (shortly *DBSCAN*) algorithm. The *DBSCAN* is a density-based clustering algorithm able to detect groups in data without the need to predefine the number of clusters [9, 24]. One of the concerns when examining Hadoop executions is detecting automatically the tasks inside an execution, so they can be properly examined. Each Hadoop task can be considered part of a *phase* of the execution (mappers and reducers), so detecting different tasks can help to see how the execution is progressing, how many resources are being used by each phase and detect bottlenecks or critical resources.

Figure 7 shows the ALOJA cross-examination tool, where tasks and phases of an execution can be observed in a scatter-plot against two metrics selected by the user. The purpose of examining tasks inside an execution is also to detect which of them behave different from the others. Where most of the tasks take about 60 seconds, while a small group of tasks take 25 seconds. Furthermore, the framework user can select the type of task to be clustered, so clustering can be done inside the same class of task *e.g.*, mappers or reducers.

Another use of the DBSCAN exploratory method (the DBSCAN-execs tool) is to compare the impact of software or hardware configurations in a given execution, by comparing directly its tasks behavior. Not like the tasks examination tool, which examines a single job tasks, the DBSCAN-execs uses all the selected executions in order to compare tasks across the different executed configurations and deployments. The DBSCAN-execs tool calculates the DBSCAN for all the available executions, keeping the centroids for each cluster found, thus such centers are the representation of the different tasks inside the execution. Then, DBSCAN is re-executed for the observed execution centers, so tasks (i.e., their representative centers) are clustered by behavior. Each final cluster represents a set of tasks that behave similar on possibly different configurations. The tool displays the centers of the obtained clusters against two metrics decided by the user. As an example, Figure 8 shows how providing different Hadoop block sizes to an execution affect the duration and bytes read per task. This visualization tool allows to compare different configurations or deployments through the performance of the execution tasks.

Fig. 8. Example of the DBSCAN-EXECS displaying the duration and bytes read for k-means benchmark executions, highlighting the detected block sizes

V. Predictive Analytics in ALOJA

The number of configuration choices as the project expands in architectures and services —in the *millions* for benchmarks that a single iteration can take *hours* to execute– continue to increase. In order to cope with the increasing number of configuration options the project was faced with the need first to do manual sampling from the search space, and grouping of results to extrapolate results between clusters. However, this initial approach has not been sufficient either, and still requires a large number of benchmarks. For this reason, we have leveraged Predictive Analytics (PA) techniques to be able to optimally execute a smaller number of benchmarks among other uses. PA encompasses a variety of statistical and ML techniques to make predictions of unknown events based on historical data [11, 28] —in this case the aggregated metadata of our benchmarking repository. Machine Learning (ML) is the part of data mining in charge of automatic modeling and prediction, or what nowadays this is known as "Predictive Analytics". Figure 9 shows the machine learning schema applied to our case of study.

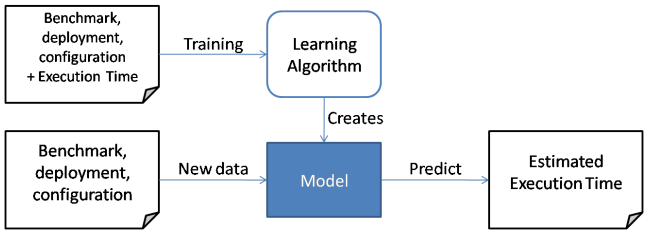

Fig. 9. Machine Learning methodology in our framework

Based on running examples from a Hadoop environment, we can attempt to obtain models of such environment, if not as good as a model built by an expert, acceptable enough [4]. At this time, when focusing on Hadoop environment modeling we rely on benchmarks, oriented to measure performance by driving the framework through different but identified states and distinct resource usage levels. We look for executions

stressing the more different parts of the system as possible and having samples representing each possible behavior. This is, when using ML algorithms to automatically model environment behaviors we want to obtain a representative sample of the space of possible configurations on Hadoop set-ups, and different benchmarks can provide us this sparsity of samples.

At this time, the ALOJA data-set has some challenging issues we have to deal with: find accurate models explaining the behavior of a deployment, a benchmark or a given configuration. The ALOJA-ML component of the platform provides algorithms for learning over the execution data-sets, also provides tools that apply the learned models to predict executions, recommend configurations and detect anomalies on introduced executions.

The executions in the ALOJA data-set, or even contribution executions to be introduced by users, are usually executed following different reasons than to obtain a homogeneously representation of every benchmark and feature. Those executions use resources and takes time to execute —thus they cost money. Also covers the whole space of possible configurations with real executions can represent a significant cost and time. One of the goals of ALOJA-ML is to be able to infer models from a not-so-regular data-set and perform *virtual executions*, this means to predict executions for unseen configurations, but treat them as if they were actual executions in other tools. In addition, introduced executions can contain failed or outlier runs, as external factors can affect the execution time in a heavy manner. Some of those examples can be easily detected and removed by simple review, while others can not.

A. Modeling Benchmark Behaviors

The modeling methodologies in ALOJA-ML are based on machine learning. The model of a system is obtained by collecting examples of this system, including the information we want to be able to predict and any element that can determine or affect it, and passing this example data-set through a learning algorithm able to infer a function describing the system behavior. This function receives as inputs variables like in our case the benchmark name, the introduced Hadoop configuration and the hardware specifications, and produces as output the required execution time for the benchmark in such conditions.

The main concern of ALOJA-ML is to model benchmark execution times to predict those in the more automatic possible way. Hadoop executions can be tuned through software configurations (number of mapping agents, type of compression used, etc.) and through the provided hardware resources. These tuning parameters determining the resulting execution are the input variables, and the resulting execution time is at this time the output variable we want to predict. And predicting such values will let ALOJA to plan benchmark executions, compare different environments without having all the configurations executed, also plan new data-centers by selecting components while predicting the execution time for each of them.

B. Learning Models

The learning tool-set follows a 3-step way to train and test models, in order to train and tune the parameters of the learning algorithms with a different data-set than the one used to evaluate the model. This is done for 1) avoiding over-fitted models, by using separate data-sets to create the model and evaluate it with unseen data, also keeping a third set for after tuning the algorithm and re-training; and 2) evaluating the model always with data not involved in the training and tuning process. Here the data-set is split in 3 parts, one for training the model, one for validating the model while tuning the algorithm, and one for testing the model. Also we incorporate in the tool-set different learning algorithms for the user to select, like regression trees (M5P algorithm [20, 27]), the k-nearest neighbors method [3], or artificial neural networks [26].

At this time we include the benchmark details, the Hadoop software configurations and the deployment hardware properties as features for our training. The selected variables input for the following experiments are *benchmark*, *network*, *storage type*, *number of maps*, *block size*, *sort factor* and *file buffer size*, *cloud provider* information and *deployment HW and OS* information; and *execution time* as output variable. As some of those variables provide more information than others, we let some variables with low relevance at user's decision, as the ALOJA framework can filter the target data-set to examine or model subsets of it where those variables can become relevant.

Our machine learning framework has been tested in previous works (see ALOJA-ML [5]), where some preliminary results with the initial data-set of 18.000 executions were presented. As the database of executions has expanded since then, also new attributes have been introduced (like *type of benchmark* {HiBench 100GB, HiBench 1TB, HiBench for HDI-Insight}), here we present some results supporting the current evolved data-set. Using a random split to create 3 data-sets (training, validation and testing) with sizes 50%, 25% and 25% respectively. Figure 10 shows the results of creating models with the three aforementioned ML algorithms.

Algorithm	MAE	RAE	Parameters
Regression Tree	222.27	0.19170	M = [2,5]
Nearest Neighbors	219.96	0.13700	K = 5
FFANNets	290.56	0.26690	32:6:1, iters 1000, decay $5 \cdot 10^{-4}$

Fig. 10. Mean and relative absolute error on best parameters found

With the current classic learning methods we can create general models incorporating all the benchmarks, with an accuracy between 0.13 and 0.20 RAE using classic methods and removing those elements that we consider trivial outliers (executions marked as failed or unfinished), and using all the data-set. But one of our concerns is to see which level of generalization a model can achieve, given that we have different benchmarks and a huge space of possible configurations. Each benchmark has different characteristics leading to different behaviors (CPU bound workloads, IO bound workloads, etc.). If a learning algorithm can discover relevant differences among benchmarks, we can train a model with all benchmark examples by tagging each example with the benchmark name. A general model would allow to perform the learning process one single time, while specific models would fit better to their benchmarks but more example executions would be required. Figure 11 shows the results from learning a general model against specific ones.

We observe that generating specific models for each bench-

Algorithm	MAE	RAE	Algorithm and Params
bayes	25.3517	0.04788	k-NN, k = 3
dfsioe_r	656.0176	0.24940	k-NN, k = 3
dfsioe_w	190.9483	0.13489	M5P, M = 1
kmeans	213.4287	0.11526	M5P, M = 1
pagerank	218.9305	0.42921	M5P, M = 5
sort	315.9926	0.24817	M5P, M = 5
terasort	169.8481	0.13238	M5P, M = 5
wordcount	146.5111	0.08147	k-NN, k = 1

Fig. 11. MAE and RAE for the best method and algorithms found per each benchmark

mark, or it fits better than the general (probably caused by over-fitting), or it degrades the result caused by low examples of that given benchmark (also in some cases it is because such benchmark has unstable executions, like in *pagerank*). If we consider targeting a specific benchmark, creating a specific model is an option. But in ALOJA we want to be able to apply our analytics to the broad range of benchmarks by reducing the required executions, avoiding repeating running a huge set of executions as sample for each one. The ALOJA-ML future work will focus on benchmark parameterization, by attempting to describe a benchmark from few executions and use these results to provide a description to be used as a substitute of their identification tag. That way, training a model from features with the characteristics of the benchmark instead of a tag or name, will ease the way to create general models even for not yet seen benchmarks or workloads.

C. Analytic Tool-set

Our framework present different analytics tools, and ALOJA-ML provide model-based enhancements like *Outlier Detection* and *Configuration Recommendation Systems*.

1) Anomaly Detection: Detecting which executions introduced in our data-set is crucial to perform comparisons and visualizations without noise and misdirection. One of the model-based enhanced tools is the Anomaly Detection mechanism, that using learned models tags as *warning* or *outlier* the introduced executions.

Considering a learned model, created through the ALOJA learning tool, as the 'system explanation', any observation that fits the model is considered *legit*, and if not it is considered *anomalous*. In such case, the tool looks for other executions coinciding with the anomalous one, deciding if this is just an unseen execution (not yet represented by the model, flagged as warning) or an outlier (clearly an execution with anomalous behavior).

The tool can be configured with the tolerance respect the execution observation and the expected behavior, also the number of recorded instances required to confirm that an anomalous execution is an outlier or just new information. Figure 12 shows an application of the tool, indicating legit versus anomalous (warnings and outliers) executions of a selected subset of executions.

2) Guided Benchmarking: When introducing a new deployment or modifying characteristics of a set-up, re-running a benchmark batch is useful to compare with previous set-ups or checking the magnitude of changes. To prevent running entire

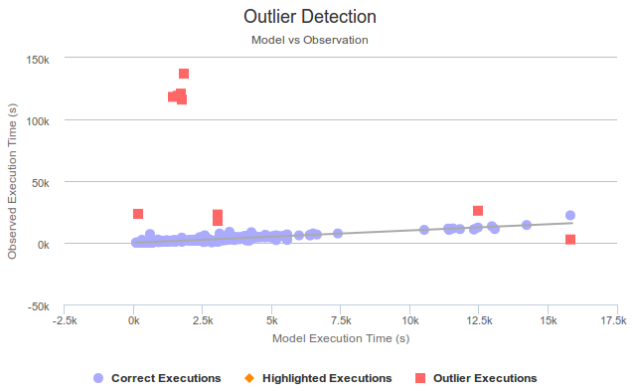

Fig. 12. Example of visualization of data-set anomalies, for Terasort executions using local disks

batches again, with their related costs, we look for repeating the benchmarking using the minimum executions but obtaining the same information to compare.

Once having a full data-set of executions, we can model it. However, we can also attempt to find the smallest subset of benchmarks them that can be used to build a similar model. Selecting the subset can become difficult, and an easy alternative is to cluster the executions by similarity, and retrieve the centers of each group as the representative of each subgroup. Our framework has available a tool to create the minimum set of executions representing selected sets of our executions, to indicate the user or an automated mechanism which executions are recommended to run in a specified known or new cluster.

The tool applies the *k-means* algorithm [21] to retrieve sets of configurations to be executed instead of re-running the entire data-set. The user can decide the number of the executions to be run, knowing that more executions will make the benchmarking process more accurate, but more expensive in time and resource costs. Figure 13 shows an example of the visualization of recommended executions for a set of clusters (on-premise and Azure) running terasort, where the chart displays the estimated error respect running all the executions against the amount of recommendations run (more runs will reduce the gap, but will cost more, and the user can chose how many recommendations want to be displayed.

3) Representing Characteristics: Finally, we introduce a tool to explore the space of configurations by using the learned models to predict all (or a subset) of Hadoop configurations and deployments, and visualize the results as a tree indicating which features (network, disk, maps, etc.) discriminate better the results. This will help to show which feature will affect more the execution time, so when choosing a configuration focus first onto the more influential. Instead of just providing a list, a tree can show which features affect more after having selected one already. Using different algorithms and indicators, like the GINI index [23] or the capacity to separate slow from fast configurations, the visualization shows the resulting tree.

Figure 14 depicts an example, where we select an on-premise cluster formed by 3 data-nodes with 12 cores per VM and 128GB RAM, and we want to observe the relevance

	Profiling	Benchmarking	Aggregation	Predictive Analytics
Information type	Debug	Speedups	Insights	Foresight
Search space	Constrained	HW and SW configs	Large	Unconstrained
Analysis	Manual	Semi-manual	Semi-automated	Automated
Precision	High	Normal	Averages	Estimations
Datasizes	Very large	Large	Small	Very small
Processing	Medium	Medium	Fast	Slow
Main focus	Application	Framework	Comparing systems and HW confs	Providers
Tolerates noise	Minimal	Minimal	Yes	Yes (depends on alg.)

TABLE II. HIGH-LEVEL COMPARISON OF THE DIFFERENT TECHNIQUES USED IN ALOJA

Fig. 13. Example of visualization of recommended executions for a cluster running Terasort and local disks

of variables *disk* (local SSD and HDD), *network* (InfiniBand and Ethernet), *IO file buffer* (64KB and 128KB) and *block size* (128, 256) for the benchmark *terasort*, fixing then the other variables (*maps* = 4, *sort factor* = 10, no compression and 1 replica). Once we have trained a model we proceed to predict all the configurations available for this scenario, and we apply a dichotomous split of variables, selecting first the one that provides less splits between fast and slow configurations. In this example, the model and tree visualization realize automatically that, for the selected range of configuration and deployments, the more discriminating feature is the Disk, in case of HDDs the network plays as second discriminator, and finally the File Buffer decides.

VI. CONCLUSIONS

This article delineates the evolution of ALOJA's focus and approach over the last year to automatically characterize Hadoop deployments and gather performance insights that can improve the cost and execution of current clusters. On our path from low-level profiling to Predictive Analytics (PA) —our current frontier, we have found specific use cases for the tested performance extraction techniques. As well as presenting the data sizes of the different techniques and the implications of handling such data. Among the results, we have found that

```
Disk=SSD
    IO.FBuf=131072 -> 970s
    IO.FBuf=65536 -> 1036s
Disk=HDD
    Net=ETH
        IO.FBuf=131072 -> 2166s
        IO.FBuf=65536 -> 2250s
    Net=IB
        IO.FBuf=131072 -> 2653s
        IO.FBuf=65536 -> 2737s
```

Fig. 14. Example of estimation of the selected space of search, with the corresponding descriptive tree

detailed performance counter collection accounts for over 99% of the data produced, while summaries give the most value to rapidly obtain cost and performance insights from the data. Aggregated summaries allow us to use PA techniques with more ease. These initial results already help us to automate, speedup and reduce costs of benchmarking efforts.

Table II summarizes and compares the different techniques currently employed in ALOJA and highlights their difference from the perspective of the project. The four techniques presented: Profiling Benchmarking, Aggregation, Predictive Analytics; are classified according to the type of information that they can provide, the amount of the search space that they can cover, the precision, focus and the automation provided. While for example Profiling produces a large volume of data, it is very accurate, but usually needs to be manually revised for each execution. While on the other end, PA works best over summarized aggregated data. Providing predictions very fast, but at a cost of training a model first and some precision loss.

Currently the ongoing PA efforts show promising results to automatically model system's behavior from existing benchmarks *i.e.*, predicting job execution times with and accuracy of up to 8% of error for specific models, or up to 13% for a general model covering the whole dataset. With the *Guided Benchmarking* feature, we can reduce the number of benchmark runs needed, by prioritizing executions according to the amount of predicting accuracy that they give. As well as for Knowledge Discovery (KD) to find relations among software and hardware components and provide recommendations for users *e.g.*, the most cost-effective VM type and cluster size in a Cloud provider. Also the expected speedup of a new disk configuration or network technology. Techniques that jointly support foresighting cost-effectiveness of new defined systems,

and importantly for the project, reducing benchmarking time and costs by using the prediction models.

Our current efforts are to integrate the PA tools to be used with the rest of the analytical tools we have implemented, as well as to increase their accuracy and use cases. For further details on the predictive analytics tools, please refer to our recent work [5], where each tool internals are explained and the methodology is described in more detail. We are also expanding the Big Data applications and frameworks, as well as to expand the number of supported systems and cloud providers. We also will like to invite fellow researchers and Big Data practitioners to download and make use of our open source tools, and to expand on the available analytic tools and contribute to the offered online benchmark repository.

ACKNOWLEDGEMENTS

This work is partially supported by the BSC-Microsoft Research Centre, the Spanish Ministry of Education (TIN2012-34557), the MINECO Severo Ochoa Research program (SEV-2011-0067) and the Generalitat de Catalunya (2014-SGR-1051). We would like to thank the *Azure4Reseach* [10] program for providing most of the cloud resources, as well as the new resources provided by Rackspace Inc [22] and the Future SOC Lab [13].

REFERENCES

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

[2] Administrator privileges on headnode of hdinsight-cluster: http://www.postseek.com/meta/bd1cddf3af9c7ce35d147e842a6864f0/1,3, May, 2015.

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

[4] 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, October 1-4, 2013*, pages 858–867, 2013.

[5] J. L. Berral, N. Poggi, D. Carrera, A. Call, R. Reinauer, and D. Green. Aloja-ml: A framework for automating characterization and knowledge discovery in hadoop deployments. In *21st ACM SIGKDD Conf. on Knowledge Discovery and DataMining, KDD 2015, Sydney, Australia*, 2015.

[6] BSC. Aloja home page: http://aloja.bsc.es/, 2015.

[7] BSC. Performance tools research group page: http://www.bsc.es/computer-sciences/performance-tools, 2015.

[8] M. A. Cloud. Vm types: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/, 2015.

[9] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[10] M. A. for Research. Project home page: http://research.microsoft.com/en-us/projects/azure/, 2015.

[11] Gartner. Predictive analytics: http://www.gartner.com/it-glossary/predictive-analytics, May, 2015.

[12] J. Guitart, J. Torres, E. Ayguad, J. Oliver, and J. Labarta. Java instrumentation suite: Accurate analysis of java threaded applications. In *PROCEEDINGS OF THE SECOND ANNUAL WORKSHOP ON JAVA FOR HIGH-PERFORMANCE COMPUTING, ICS00*, pages 15–25, 2000.

[13] Hasso-Plattner-Institut. Hpi future soc lab: http://research.microsoft.com/en-us/projects/azure/, 2015.

[14] D. Heger. *Hadoop Performance Tuning https://hadoop-toolkit.googlecode.com/files/White paper-HadoopPerformanceTuning.pdf*. Impetus, 2009.

[15] D. Heger. *Hadoop Performance Tuning - A Pragmatic & Iterative Approach*. DH Technologies, 2013.

[16] G. L. N. B. L. D. F. B. C. S. B. Herodotos Herodotou, Harold Lim. Starfish: A self-tuning system for big data analytics. In *In CIDR*, pages 261–272, 2011.

[17] 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 International Conference on*, 0:41–51, 2010.

[18] K. Kambatla, A. Pathak, and H. Pucha. Towards optimizing hadoop provisioning in the cloud. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[19] 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 *2014 IEEE Intl. Conf. on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, pages 905–913, 2014.

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

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

[22] I. Rackspace US. Company home page: http://www.rackspace.com/, 2015.

[23] L. Raileanu and K. Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, 2004.

[24] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Min. Knowl. Discov.*, 2(2):169–194, June 1998.

[25] B. Schwartz, P. Zaitsev, and V. Tkachenko. *High Performance MySQL*. O'Reilly media, 2012.

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

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

[28] Wikipedia. Predictive analytics: http://en.wikipedia.org/wiki/predictive_analytics.

[29] Z. Zhang, L. Cherkasova, and B. T. Loo. Optimizing cost and performance trade-offs for mapreduce job processing in the cloud. In *Network Operations and Management*

Symposium (NOMS), 2014 IEEE, pages 1–8. IEEE, 2014.