2018 IEEE 34th International Conference on Data Engineering

Dione: A Framework for Automatic Profiling and Tuning Big Data Applications

Nikos Zacheilas, Stathis Maroulis, Thanasis Priovolos, Vana Kalogeraki
Department of Informatics
Athens University of Economics and Business
Athens, Greece
Email: {zacheilas, maroulise, priovolos, vana}@aueb.gr

Dimitrios Gunopulos
Department of Informatics
and Telecommunications
University of Athens
Email: dg@di.uoa.gr

***Abstract*—In this demonstration we present *Dione* a novel framework for automatic profiling and tuning big data applications. Our system allows a non-expert user to submit Spark or Flink applications to his/her cluster and Dione automatically determines the impact of different configuration parameters on the application's execution time and monetary cost. Dione is the first framework that exploits similarities in the execution plans of different applications to narrow down the amount of profiling runs that are required for building prediction models that capture the impact of the configuration parameters on the metrics of interest. Dione exploits these prediction models to tune the configuration parameters in a way that minimizes the application's execution time or the user's budget. Finally, Dione's Web-UI visualizes the impact of the configuration parameters on the execution time and the monetary cost, and enables the user to submit the application with the recommended parameters' values.**

I. Introduction

In this demonstration we present *Dione*, our framework for profiling and tuning big data applications that run on distributed systems like Apache Spark[1] and Apache Flink[2]. *Dione* was developed in the context of the Vavel EU-funded project[3], to efficiently monitor and optimize the execution of Spark and Flink applications that analyze voluminous urban data coming from multiple heterogeneous input sources (*e..g.,* bus sensors, CCTV cameras) [1].

Recent works [2], [3] have shown that system parameters such as the number of worker instances and the allocated memory can greatly affect the execution time of the applications and cause significant degradation in their performance, if not properly configured. Therefore, it becomes imperative to appropriately tune these parameters to avoid penalizing the application's performance. However, the tuning procedure can be a cumbersome task for non-expert users as they are typically familiar with specific systems (such as Hadoop[4] or Spark), lack knowledge of the underlying system architecture [3] and auto-tuning frameworks like [2] rely on the availability of historical data to use in the tuning process.

The problem becomes more challenging when applications execute on public cloud infrastructures like Amazon's EC2[5]. In such environments users are charged on a per hour basis based on the amount of nodes they reserve. So apart from the application's execution time, the monetary cost is another metric that should be considered when tuning the configuration parameters and submitting big data applications in the cluster [4]. For example, we expect that users in such environments would want to know (a priori to the application submission) the appropriate parameters' values (*e.g.,* how many Virtual Machines to reserve) so that they better achieve their objective (expressed as minimizing the monetary cost or the applications' execution times).

Dione is a framework that tackles the aforementioned issues and is able to efficiently profile and tune the configuration parameters of big data applications that have been implemented for big data processing systems (such as, Apache Spark and Apache Flink). The main contributions of *Dione* are the following:

- *Dione* utilizes prediction techniques [5] to estimate the impact of basic configuration parameters such as the number of reserved nodes on the applications' execution time and spending budget. It further minimizes the number of the necessary profiling runs for building an accurate prediction model exploiting the Bayesian optimization technique [6] .
- It is the first framework that exploits similarities between the execution plans of already executed applications and newly submitted ones. The idea is to utilize an already trained prediction model for the new application, if possible, and thus avoid the time-consuming procedure of building a new prediction model [7]. To compute the similarity between the applications' execution plans (which can be seen as a Direct Acyclic Graph of processing operations) *Dione* measures the Graph Edit Distance (GED) metric using a well-known approximation technique [8].
- *Dione* determines the configuration parameters to meet the user's objective (*i.e.,* minimize the user's spending budget or the application's execution time) using a Hill climbing algorithm that we have developed [3]. The user is then prompted to submit the application with one of these suggested configurations.
- Finally, *Dione* provides a rich Web-UI that enables the user to submit Spark and Flink applications to his/her

[1] http://spark.apache.org/
[2] https://flink.apache.org/
[3] http://www.vavel-project.eu/
[4] http://hadoop.apache.org/
[5] http://aws.amazon.com/ec2/

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00195

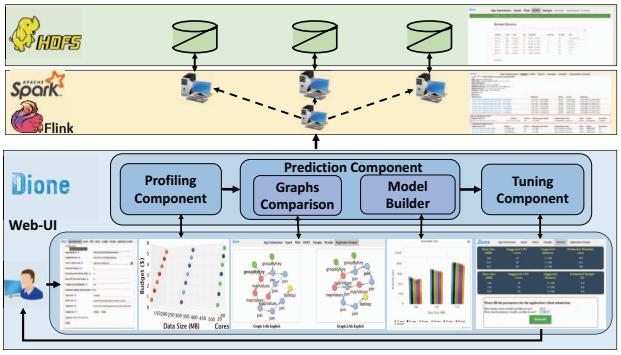

Fig. 1. *Dione*'s high level overview.

cluster. For monitoring the applications we utilize the APIs provided by the two distributed systems we support (*i.e.,* Spark and Flink) and also the Ganglia monitoring system[6] which provides lower granularity metrics like the CPU and RAM usage in the cluster. Furthermore, *Dione's* Web-UI provides a wide variety of diagrams that illustrate the effect of the tuning of the configuration parameters on the metrics of interest and also displays the parameters' values recommended to the end-user via our tuning algorithm [3].

II. System Description

A. System Overview

As we illustrate in Figure 1, *Dione* is implemented as middleware between the end-user and a Spark or Flink cluster. We consider big data applications where the data to be processed are stored in a distributed filesystem like Hadoop's Distributed File System (HDFS). *Dione* consists of three main components and a *Web-UI* for providing the results to the end-user.

Dione works as follows. The user submits through the *Web-UI* the application he/she is willing to run (*e.g.,* the jar containing the implementation of the application). Then *Dione* is responsible to examine the impact of the number of reserved CPU cores, the allocated memory per node and the input data size on the application's execution time and monetary cost by building the appropriate prediction model. Afterwards *Dione* tunes the parameters and informs the user via the *Web-UI* about the recommended parameters' values.

The *Profiling Component* monitors the execution of submitted applications on the Spark or Flink cluster. It retrieves the application's execution times and the configuration parameters that are utilized using the monitoring APIs provided by the two frameworks and exploits the Ganglia monitoring system for gathering lower granularity metrics like the nodes' CPU and RAM usage. The monitor results are used by the *Prediction Component* to estimate the applications' execution times and spending budget for different configuration parameters. The *Prediction Component* comprises two sub-components, the *Graphs Comparison* and the *Model Builder* sub-components. The main idea is to either exploit prediction models that we have already built for applications that perform similar processing (*e.g.,* applications that implement different variations of the alternative least squares recommendation algorithm) using the *Graphs Comparison* component, or determine the profiling runs (*i.e.,* using different combinations of the parameters we consider) using the *Model Builder* component that should execute in the Spark or Flink cluster in order to build an accurate prediction model. After the model has been created we use it to estimate the execution time and the spending budget for all the different combinations of the examined parameters and visualize the results using the *Web-UI* component (as we illustrate in Figure 1). Finally, *Dione* uses the *Tuning Component* to adjust the application's configuration parameters to meet the user's objective (*i.e.,* minimize the spending budget or the application's execution time). More specifically, the *Tuning Component* exploits the prediction model that has been created by the Prediction component to determine the appropriate parameters' values and the user is informed about them via the *Web-UI*.

B. Implementation Details

Graphs Comparison. Whenever a Spark or Flink application is submitted to *Dione* we retrieve its execution plan exploiting the corresponding API. The application's execution plan is a directed acyclic graph (DAG) that consists of all the operations (*i.e.,* actions and transformations as defined in Spark terminology and dataset transformations in Flink terminology) that are performed. The *Graphs Comparison* component is responsible to compare the newly submitted application's execution plan with plans we have gathered for previously executed applications. The idea is to find the most similar execution plan and exploit an already trained prediction model, if such a model exists, thus avoiding the costly procedure (both in terms of time and monetary cost) of building a new prediction model.

Our goal in the *Graphs Comparison* component is to exploit the fact that different applications have similar execution graphs and this similarity in the execution order of the processing operations can lead to similar execution times. To compute the similarity between two graphs we use the Graph Edit Distance (GED) [8] metric as it is the most appropriate measure for depicting the distance between graphs. GED is measured as the minimum amount of required distortions to transform one graph into the other. The computation of GED has exponential complexity as the problem of measuring the graph edit distance is NP-hard [8]. Therefore, the direct computation of the GED metric will take unacceptable amount of time for large graphs. For this reason we use an approximation technique that is able to approximate the GED between two application graphs in polynomial time [7].

Model Builder. In the case that the *Graphs Comparison* component is unable to find a similar previous execution plan (*i.e.,* distance smaller than a threshold) then we have to build a new prediction model for the submitted application. The *Model Builder* component is used for solving this problem. In order to build accurate prediction models we need to execute profiling

[6] http://ganglia.sourceforge.net/

runs and gather the necessary training data. However, profiling runs cost both in terms of money and time. In *Dione* we cope with the issue of gathering training data by minimizing the number of combinations tested for building the model, so that we reduce the overhead of the training phase in terms of its duration. More formally, let $f(\vec{x}), \vec{x} \in X$ be the function that depicts the application's execution time, $\vec{x}$ is the vector of the configuration parameters (*i.e.,* currently, we consider the number of reserved CPU cores, the memory and the input data size as our configuration parameters) and X is the set containing all the possible combinations of the configuration parameters that can be evaluated. Our goal is to determine a set $X^{'} \subset X$ that we will use as training dataset for the prediction model that will approximate $f(\vec{x})$.

We decided to use the Bayesian optimization technique [6] in our *Model Builder* to solve this problem. The main idea of this approach is to incrementally build a probabilistic model that reflects the current knowledge of the objective function (in our case the execution time of the application) until some convergence criterion is met. In our case we decided to use a Gaussian process as our probabilistic model as we have efficiently applied it for estimating the latency of stream processing applications in our previous work [5]. The Bayesian optimization algorithm executes three steps in each iteration. First, it performs a numerical optimization to find a point in the parameters' space which maximizes an *acquisition function*. We use the *expected improvement* function which returns the expected value of the improvement brought by evaluating $f(\vec{x})$ over the best value n found so far:

$$a(\vec{x}) = E(max(0, f(\vec{x}) - n)) \quad (1)$$

We find the $\vec{x'}$ point which maximizes $a(\vec{x})$ using a numerical optimization algorithm. After we have found this $\vec{x'}$ point, we measure the execution time of the application when the $\vec{x'}$ parameters are utilized and then update the Gaussian process model. We stop the algorithm's iterations when we have reached the convergence criterion which in our case is a user-defined upper limit on the profiling phase's execution time.

Tuning Component. For determining the appropriate parameters (*i.e.,* the number of CPU cores and the per worker memory) that should be utilized by an application we apply a greedy Hill climbing algorithm that we have previously utilized for tuning the reduce tasks of MapReduce jobs [3]. For minimizing the application's execution time the idea is to initially start with the minimum resources and gradually increase them by a step size (*i.e.,* by adding one CPU core at each iteration) until we do not observe any improvement. For the budget minimization we follow the inverse procedure we start with the case that all resources have been reserved and then gradually decrease them as long as we observe a decrease in the user's spending budget.

III. Demonstration Plan

In this demonstration, users will be able to interact with *Dione* through its web interface. More specifically, we will be using a laptop in order to demonstrate the front-end of *Dione*

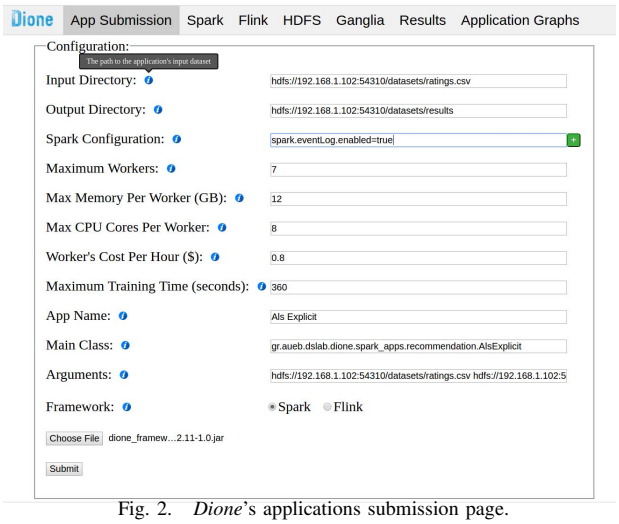

Fig. 2. *Dione*'s applications submission page.

and our 8 nodes Spark and Flink cluster (*i.e.,* Spark 2.0.2, Flink 1.3.1) for running the users' applications. The cluster consists of 1 master node and 7 worker nodes. Each node is equipped with 8 CPU processors (*i.e.,* Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz) and 16 GB RAM. The master node also hosts the Apache server that is necessary for running *Dione's* Web-UI which was implemented using a number of *Javascript* libraries.

Users will be able to submit applications to our cluster via our submit applications page (*i.e.,* see Figure 2). More specifically, the users will provide the input and output path of their application, the jar containing the application's implementation and also the program's entry point (*i.e.,* the main class). Furthermore, the users will specify the maximum number of workers in the cluster and the CPU cores/memory per worker. These parameters will enable *Dione* to determine the range of the configuration parameters that will be examined. Furthermore, the user will have to to indicate whether the submitted application will be a Spark or a Flink application.

Once the application has been submitted to *Dione*, the user has to wait for the *Prediction Component* to run and create a prediction model for estimating the execution time and spending budget of the submitted application. When the prediction model has been built, the user will be transferred to the results page where he/she can observe the impact of the different parameters on the two metrics of interest. *Dione's* Web-UI provides both $3D$ (*i.e.,* Figure 3) and $2D$ (*i.e.,* Figure 4) diagrams so that the user can observe and understand the impact of the CPU cores, memory and input data size on the spending budget and the application's execution time. In Figures 3 and 4 we display the results when an alternative least squares (ALS) application is submitted to *Dione*. Furthermore, as can be observed in Figure 5, *Dione* will suggest the configuration parameters (*e.g.,* the number of CPU cores and memory) that minimize either the spending budget or the application's execution time (*i.e.,* using the *Tuning Component* we described in Section II-B).

At the bottom of the results page, as we illustrate in

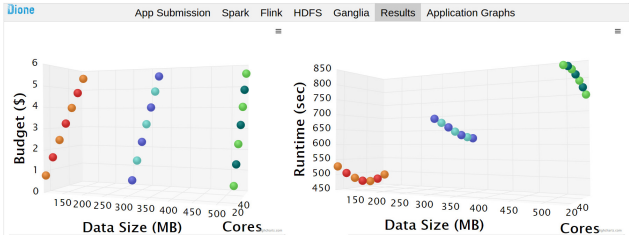

Fig. 3. Interactive 3-D plots that can be used for studying the impact of the configuration parameters on the metrics of interest.

Fig. 4. Impact of the reserved CPU cores on the execution time and spending budget for varying input data size.

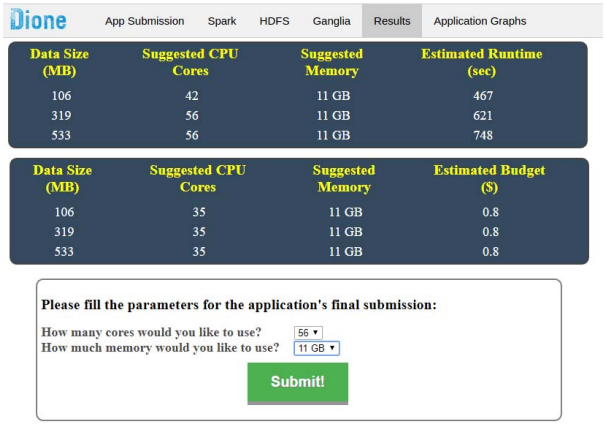

Fig. 5. *Dione's* suggested parameters for varying input data size.

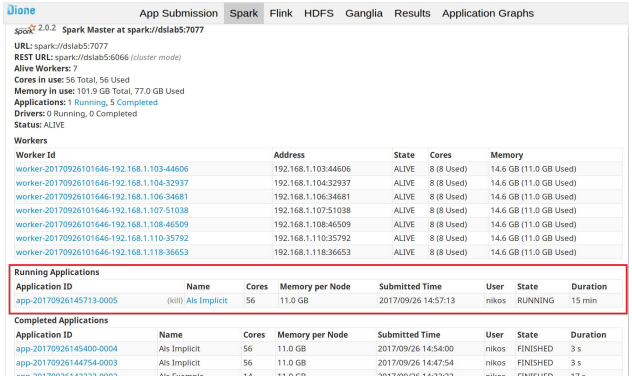

Fig. 6. Application that runs in the Spark cluster with the chosen parameters.

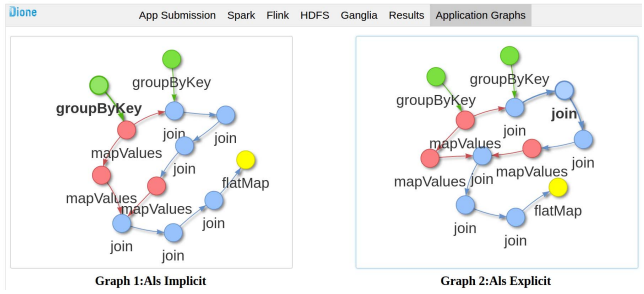

Fig. 7. Similar application graphs detected by *Dione*.

Figure 5, the user will be prompted to submit the application to the cluster with his/her chosen parameters. Afterwards, he/she will be able to observe the application's execution in the Spark (or Flink) cluster by clicking the corresponding button in the Web-UI (see Figure 6). Furthermore, we also have a link to the Ganglia monitoring framework so that the user can easily examine the resource usage (*e.g.,* CPU utilization, I/Os) in the cluster during the application's execution. Finally, when an application with a similar execution plan has been detected by the *Graphs Comparison* component then the user will see the similar operations of the two graphs, by clicking the Application Graphs button. In this web page, as we highlight in Figure 7, the user can see a subset of the processing operations (*e.g.,* map transformations) that occur in both graphs. In Figure 7 we illustrate the similarity in the graphs of two Spark applications (*i.e.,* two variations of the alternative least squares recommendation algorithm). As can be observed, the two applications have the exact same set of operations therefore the same prediction model is used.

ACKNOWLEDGMENT

This research has been financed by the European Union through the FP7 ERC IDEAS 308019 NGHCS project, the Horizon2020 688380 VaVeL project and a 2017 Google Faculty Research Award.

REFERENCES

[1] N. Panagiotou, N. Zygouras, I. Katakis, D. Gunopulos, N. Zacheilas, I. Boutsis, V. Kalogeraki, S. Lynch, and B. O'Brien, "Intelligent urban data monitoring for smart cities," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Riva del Garda, Italy: Springer, 2016, pp. 177–192.

[2] 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*, vol. 11, Asilomar, CA, USA, 2011, pp. 261–272.

[3] N. Zacheilas and V. Kalogeraki, "A pareto-based scheduler for exploring cost-performance trade-offs for mapreduce workloads," *EURASIP Journal on Embedded Systems*, vol. 2017, no. 1, p. 29, 2017.

[4] ——, "Chess: Cost-effective scheduling across multiple heterogeneous mapreduce clusters," in *ICAC*. Wurtzburg, Germany: IEEE, 2016, pp. 65–74.

[5] N. Zacheilas, V. Kalogeraki, N. Zygouras, N. Panagiotou, and D. Gunopulos, "Elastic complex event processing exploiting prediction," in *BigData*. Santa Clara, CA, USA: IEEE, 2015, pp. 213–222.

[6] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, Lake Tahoe, Nevada, USA, 2012, pp. 2951–2959.

[7] N. Zacheilas, S. Maroulis, and V. Kalogeraki, "Dione: Profiling spark applications exploiting graph similarity," in *BigData*, Boston, USA, 2017.

[8] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance," *VLDB*, vol. 2, no. 1, pp. 25–36, 2009.