PANIC: Modeling Application Performance over Virtualized Resources

Ioannis Giannakopoulos*, Dimitrios Tsoumakos§, Nikolaos Papailiou* and Nectarios Koziris*
* *Computing Systems Laboratory, School of ECE, National Technical University of Athens, Greece*
{ggian, npapa, nkoziris}@cslab.ece.ntua.gr
§ *Department of Informatics, Ionian University, Corfu, Greece*
dtsouma@ionio.gr

***Abstract*—In this work we address the problem of predicting the performance of a complex application deployed over virtualized resources. Cloud computing has enabled numerous companies to develop and deploy their applications over cloud infrastructures for a wealth of reasons including (but not limited to) decrease costs, avoid administrative effort, rapidly allocate new resources, etc. Virtualization however, adds an extra layer in the software stack, hardening the prediction of the relation between the resources and the application performance, which is a key factor for every industry. To address this challenge we propose *PANIC*, a system which obtains knowledge for the application by actually deploying it over a cloud infrastructure and then, approximating the performance of the application for the all possible deployment configurations. The user of *PANIC* defines a set of resources along with their respective ranges and then the system samples the deployment space formed by all the combinations of the resources, deploys the application in some representative points and utilizes a wealth of approximation techniques to predict the behavior of the application in the remainder space. The experimental evaluation has indicated that a small portion of the possible deployment configurations is enough to create profiles with high accuracy for three real world applications.**

***Keywords*-application profiling; application performance; cloud applications; performance modeling**

I. Introduction

Cloud Computing has brought forth a new computing paradigm, in which virtualized resources can be allocated and freed on demand. Cloud-based deployments offer multiple advantages including (but not limited to) decreased costs, less administrative burden, bigger flexibility, seemingly infinite resources to harness on a pay-as-you-go manner. As a direct consequence, almost all every-day users are using at least one cloud-based application, while the amount of businesses that take advantage of the Cloud's offerings are ever increasing [1]. This trend is particularly observed in modern compute and data intensive platforms (e.g., Hadoop, NoSQL DBs [2]), which are now the basis of every analytics application/job. As most of these engines are designed to scale horizontally, deploying them over virtualized infrastructures seems like a natural fit.

Predicting application performance is a well-known research and business problem ([3], [4], [5], [6]). Building a reliable model of application behavior offers engineers and analysts a wide range of advantages: Most importantly, it allows for better resource allocation both at deployment and during runtime. This translates to happier customers (minimizing delays, downtimes, denial of service, etc) and better cost management.

As applications become more complex, so does building accurate models of their behavior. This issue is exacerbated by the fact that many applications are now deployed over cloud infrastructures. Applications now run over virtualized resources that: 1) are highly heterogeneous between different providers and inside a single provider; 2) are shared with an unknown number of other applications; 3) have performance that is abstracted from the underlying physical layer that the user understands. As a consequence, performance varies greatly among different providers, different deployments or even different times of day. This is especially true for resource-demanding platforms that require large amounts of CPU and memory/disk I/O and scale horizontally at runtime. These engines are heavily utilized nowadays for storing and analyzing Big Data for almost any conceivable reason.

In this work, we present *PANIC* (**P**rofiling **A**pplications **I**n the **C**loud), an application profiling system that focuses on this important element: It models performance based on the amount and type of virtualized resources allocated to the application. The main goals of *PANIC* are: (i) to provide a generic methodology, applicable for any kind of cloud application deployed, (ii) to offer customizable tradeoff between performance accuracy and profiling cost and (iii) to be customizable, in the sense that a user can choose from a pool of resources for which the profiling will be executed.

The main idea of *PANIC* is to provide predictions for the application performance by actually deploying the application in representative resource combinations and approximate the performance for the rest, non deployed combinations. The system confronts the application performance, which should be a countable quantity, as a high dimensional function and it utilizes a set of approximation techniques (from regression to machine learning classifiers) to identify the behavior of the application. Since the possible combinations of resources grows exponentially with the complexity of the application, *PANIC* exploits sampling techniques in order to pick representative configurations, deploy the application according to them and, eventually, provide predictions for all the possible configurations. Through our experimental evaluation where we deploy three typical cloud applications, we demonstrate that our system is capable of providing an accurate application profile by deploying only 10% of the possible deployment setups.

II. The problem and some assumptions

Let us assume a two-tier application, consisting of a web server and a database server, deployed over an IaaS provider. We also assume that we can predict the application load and we can deploy each tier in the following possible setups:

Table I: Possible setups of a typical Web Application

Tier 1 - Web server	RAM (MB)	512, 1024, 2048
	Cores	1, 2, 4, 8, 16
Tier 2 - Database Server	Storage (GB)	5, 10, 20, 30, 100

We assume that both the Web and Database Servers will run in a single host each. The following question arises: what performance will the application achieve for different choices of the offered resources for each tier, for a specific load? Answering this question leads to an accurate performance profile of the application that, in turn, delineates the application behavior in general.

In the general case, assume that we have an application described by n dimensions. Each dimension is denoted as $d_i, i \in [1, n]$ and it can be related to exactly one application Tier. The Cartesian product of the dimensions forms the space of the possible setups (denoted as D): $D = d_1 \times d_2 \times ... \times d_n$. The performance of the application is a countable size indicating the ability of the application to fulfill its objectives (e.g. achieved throughput, latency, etc.). The performance space is denoted as P and it is a single dimensional space (our work is easily extensible to support multidimensional performance spaces). Hence, the profile of the application (denoted as p) is defined as a function $p : D \rightarrow P$, indicating the achieved performance for each acceptable deployment setup.

Since the function p is unknown and it cannot be estimated for the entire input space, we address its estimation as a typical function approximation problem. Specifically, we want to estimate the function $\widehat{p} : D \rightarrow P$ with respect to keeping $\sum\limits_{d \in D} |\widehat{p}(d) - p(d)|$ minimum. The approximation process involves sampling D (let D_s represent the set of sampled points) and calculating the values $p(d) \forall d \in D_s$. D_s, along with the respective values $p(d_i), d \in D_s$ are given as input to our approximation algorithms, which in turn create the function $\widehat{p}$. We utilize a large number of approximation techniques, from regression to machine learning and classification algorithms. Since the estimation of $p(d)$ entails the actual deployment of the application, it is obvious that the needed time to estimate $\widehat{p}$ is dominated by $|D_s|$: assuming that the deployment time is constant regardless of the deployment setup, the number of deployments will eventually determine the execution time. Furthermore, the points that are going to be picked into D_s have a huge impact on the accuracy of $\widehat{p}$.

III. Our approach

In Algorithm 1, the general methodology used for creating a profile for a given application is provided. The algorithm expects a valid application description A followed by an input domain D, representing the possible setups the application can be deployed into and a list of applicable models. The profiling process occurs iteratively: while the termination condition is not fulfilled, the domain space is sampled, a new point p is picked and the application is deployed according to p. The deployment produces a performance metric d which is then used to train in an incremental manner all the available models. The output of the process is the model which achieves the highest accuracy, according to a user specified metric.

Algorithm 1 Main profiling algorithm

```
Require: application $A$, input domain $D$, models $M$
Ensure: model $m$
while not termination_condition do
    $p \leftarrow$ SAMPLE($D$)
    $d \leftarrow$ DEPLOY($p$)
    for m $\in M$ do
        m.train_incrementaly($p$,$d$)
    end for
end while
return best_model($M$)
```

The termination condition can vary. In many cases, it can be a threshold of sampled points that, if reached, the condition is true and the algorithm terminates. In other cases, it can be related to the achieved accuracy: if the trained model achieves to predict the objective function with error lower than a user defined threshold, the termination condition is reached. As we will present in the following section, the nature of the termination condition directly relates to the nature of the sampling algorithm.

A. Sampling

The sampling procedure occurs at the beginning of each profiling loop. The sampler receives as input the domain space D of the application, which is composed of all the acceptable deployment points. If the termination condition in Algorithm 1 is related to the number of sampled points, then the sampler receives as input a positive number $0 < s \leq 1.0$ indicating the maximum number of points that the sampler should return, as a portion of the number of points in D. Each point returned by the sampler will be used for deployment, the application performance will be measured and then an approximation model will be trained using the acquired information.

There are many methodologies for sampling a multidimensional space; We can categorize the methods we support in the following categories: (i) Static sampling, where the sampler needs no other information than the domain space characteristics (dimensions and acceptable values) to pick the next sample, (ii) Adaptive sampling, where the sampler exploits the knowledge obtained by the deployment of previously picked samples.

The static approach does not take into consideration the application performance. Typical examples of static sampling are the *Random* sampler, that returns random points and the *Uniform* sampler which constructs a multidimensional grid in the input space D, and returns points belonging to the grid. The adaptive approach, on the other hand, exploits the knowledge obtained from each deployment/sample, enabling the sampler to return more samples in regions of the domain space D where the performance appears to have fluctuations. Equivalently, an adaptive sampler will favor areas of D that affect the application performance more. In Algorithm 2 we provide the *Greedy Adaptive Sampling Algorithm*.

Algorithm 2 Greedy Adaptive Sampling Algorithm

```
Require: input domain $D$, chosen samples $L$, number $K$
Ensure: sample $s$
if $|L| < K$ then
    $s$ = borderPoint($D$)
else
    max = 0
    for all $t_1 \in L$ do
        for all $t_2 \in L$ do
            $a$ = find_midpoint($t_1$, $t_2$, $D$)
            if $|t_1 - t_2| > max$ and $a \notin L$ then
                max = $|t_1 - t_2|$
                $s$ = $a$
            end if
        end for
    end for
end if
return $s$
```

The algorithm expects as input the domain space D, the list of all the previously picked samples L and a positive number K. At first the algorithm returns K points from the border of the hyperplane defined by all points $d \in D$. The border points are picked with the notion that if the application is deployed using the highest and lowest available resources for each dimension (in combination, between different dimensions), the objective function will most likely present its highest and lowest values in the respective points. Furthermore, the K border points are also equidistant in order to avoid high gathering of points in a small region of the input space. When the number of chosen points exceeds K, the algorithm then utilizes the knowledge obtained from the first samples. Specifically, the distances[1] between all the points are calculated and the midpoint between each couple is estimated. The midpoint is defined as follows: assuming 2 points $p_1, p_2 \in D$, the midpoint p_{med} is the point whose values for each dimension equal to the average values of points p_1, p_2 to the respective dimensions. If such point does not exist (e.g. such resource combination is not applicable), the geometrically closest point is returned. The eventually picked midpoint is the result of the most distant points, as long as this point was not previously picked.

B. Approximation models

When a new sample is picked by the sampler and deployed, the performance metric for the deployment is stored and given as input to an approximation model. The training set of the model consists of the chosen samples along with their performance values. After the training process is finished, the model will be able to approximate the objective function for the entire space D.

There exist many methodologies for approximating an unknown function. We can categorize them in two major categories: regression based techniques and classification techniques. Algorithms on the former category create an analytical form of the objective function. The classification techniques, on the other hand, do not create an analytical function but rather classify the points of the domains space in classes; These objects are treated in a similar manner, indicating that the same properties stand for objects in the same class.

In our approach, we utilize the approximation models offered by *WEKA*[7], an open source data mining software which implements a variety of machine learning algorithms. WEKA provides a handful of approximation models including, but not limited to: (i) Multilayer Perceptron, that represents a typical neural network with many hidden layers and neurons, (ii) Linear Regression (Least Median Squares), that implements the methodology introduced at [8], (iii) RBF Network, which trains a Radial Basis Function Network, as presented at [9], (iv) Gaussian Process, that approximates the objective function using gaussian distributions, etc.

The accuracy of each of the models is highly affected from its configuration and the nature of the objective function. For example, a linear hyperplane will be approximated faster using a linear regression method; On the contrary a complex surface which has spikes and valleys is more likely to be approximated more accurately using a non linear approach.

[1] The points t_1 and t_2 represent points $p_1, p_2 \in D$ along with their respective performance values r_1, r_2, so $t_1 = (p_1, r_1)$ and $t_2 = (p_2, r_2)$. The norm $|t_1 - t_2|$ in this paper represents the difference $|r_1 - r_2|$, until otherwise stated.

All the available models are trained in parallel by the system, and the most accurate model is eventually picked.

C. System Architecture

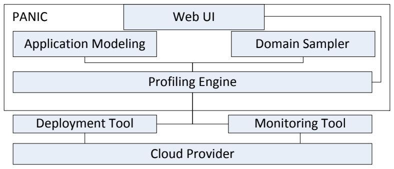


Figure 1: The architecture of PANIC

In Fig 1 we provide the architecture of the system. The core component of our system is the *profiling engine*. It is responsible for the synchronization between different tasks and it orchestrates the different components to achieve the common goal. Each time a new profiling loop is triggered, new application models and a new domain sampler is initialized (according to the user preferences). The sampler will initially create requests for new deployments and the Profiling Engine will forward the request to the Deployment Tool (a tool written for the needs of PANIC). When the execution of the application is terminated, the monitoring tool collects the user specified performance metrics and forwards them to the Profiling Engine. The monitoring system which is used by default is Ganglia[10]. The engine will then retrain the previously initialized Application Models and an accuracy estimation will occur. If the desired accuracy is achieved, then the profiling process is terminated, else a new profiling iteration occurs. The whole process is exported to the user through a Web UI.

IV. Experimental evaluation

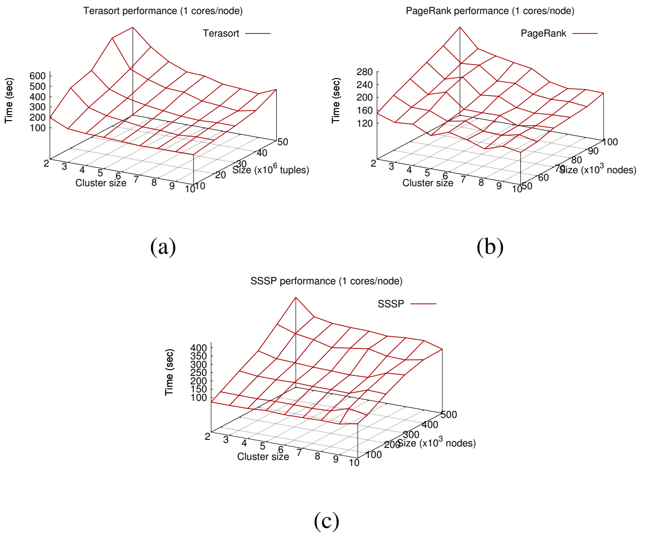
To evaluate the performance of *PANIC*, we have selected a set of distributed analytics jobs/applications that are naturally deployed over large scale virtualized resources. The first benchmark application is TeraSort [11], a well-known benchmark that sorts a set of key values. We test it with datasets of 10M up to 50M key-values (1GB to 5GB of data respectively) and run the TeraSort in Hadoop clusters with different number of nodes and different number of cores per node. The second application is a BSP-based implementation of PageRank [12], a well known graph algorithm implemented over the Apache Hama framework. We utilize 50K to 100K node graphs, each of which has at most 50 outgoing edges and execute PageRank over different cluster sizes as above. Finally, the third application is a BSP implementation of the Single Source Shortest Path (SSSP) algorithm [13], implemented for the Apache Hama framework. For SSSP, we create synthetic graphs consisting of 50k up to 500k vertices and at most 50 edges per node. For all the aforementioned algorithms, the performance metric we seek to predict is execution time.

The domain space for each of the three applications is composed of two dimensions related with the virtualized resources and one dimension related to the application load which, in our case, is intimately related to the size of the input dataset. The dimensions along with their respective values are provided in Table II. To evaluate the efficacy of *PANIC*, all three applications have been deployed for each possible combination. Consequently, the sampling algorithms presented in the previous section were applied and classifiers were trained, allowing us to measure the prediction accuracy.

Table II: Resource Dimensions

Dimension	Values	
Nodes	2, 3, 4, 5, 6, 7, 8, 9, 10	
Cores/node	1, 2, 4	
Dataset size	Terasort (Millions of Key Values)	10, 20, 30, 40, 50
	PageRank (Thousands of Nodes)	50, 60, 70, 80, 90, 100
	SSSP (Thousands of Nodes)	50, 100, 200, 300, 400, 500

A. Raw performance



(a)

(b)

(c)

Figure 2: Raw performance

The running times for all three benchmark applications is given in Figure 2. We only provide the execution times of each benchmark application for the single core VM cases; The 2 and 4 cores cases are not provided due to space constraints. In Figure 2a we provide the execution time of the Terasort benchmark with regard to the size of the cluster and the dataset size (measured in millions of key-values). It is obvious that the execution time is inversely proportional to the cluster size and proportional to the dataset size. Furthermore, for large clusters we notice that the execution time decreases less rapidly, because the communication overheads affect more the overall execution time.

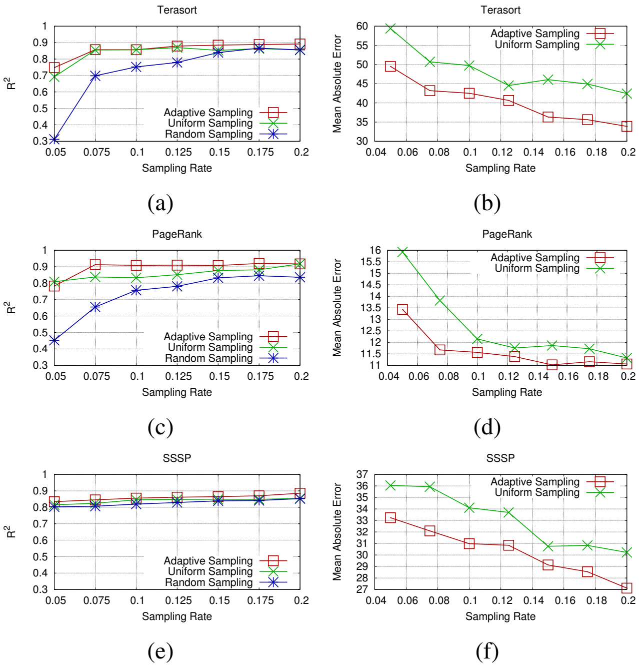
The execution time for both PageRank and SSSP are also shown in Figures 2b and 2c respectively. PageRank has a similar behavior to the Terasort case. SSSP, on the other hand, presents a slightly different behavior in terms of scalability. Specifically, when more nodes are added to the Hama cluster, the execution time remains unaffected for smaller dataset sizes (e.g., 50k nodes), while for larger datasets it decreases, but less rapidly than in the other cases. This is due to the larger number of supersteps executed by SSSP. Specifically, for our datasets, each SSSP job requires about 25–30 Hama supersteps while PageRank requires only a third of them. As a consequence, SSSP needs more sequential steps thus more time for synchronization between the BSP workers. Thus, due to this cost, the addition of more workers does not greatly benefit SSSP.

B. Sampling rate

One of the greatest factors that affect the performance of our system is the *sampling rate*. This is defined as the ratio between the number of the chosen points and the total number of acceptable deployments. Lower sampling rates lead to fewer chosen points, offering the classifiers less knowledge for the objective function (the performance of the application). Via the coefficient of determination R^2 [14] we quantify the accuracy of the profiling methods. R^2 declares the degree in which a classifier fits the original data. It is calculated as follows: $R^2 = 1 - \frac{\sum\limits_i (y_i - f_i)^2}{\sum\limits_i (y_i - \overline{y})^2}$ where y_i are the real performance values, f_i are the predicted values and $\overline{y}$ is the mean of the observed data. The closer R^2 gets to 1.0, the better the performed approximation. We also utilize the Mean Absolute Error [15] metric which is defined as: $MAE = \frac{1}{n}\sum\limits_i |f_i - y_i|$.

We applied the sampling methodologies presented in Section III-A and trained all the available models of Section III-B with the chosen points along with the respective performance values for different sampling rates. In Figure 3 we provide the accuracy level of the best model for each sampling rate for all three applications; In Figures 3a, 3c, 3e, R^2 is depicted whereas the ones on the right (3b, 3d, 3f) represent the *MAE*. The best model is defined as the model that presents the highest coefficient of determination. The respective deviation for each application did not overtake 10% of the values of MAE and we do not depict it on the figures.

In our results, we notice that the most accurate models present slightly different behavior for each one of the three applications. In all of them, it is obvious that an increase in the Sampling Rate leads to higher accuracy. This result is expected, since higher sampling rate means that more points are picked, thus the model will obtain more knowledge for the objective function. However, in many cases this might not be the case: The sampler may pick more points but if they are not representative ones, they may mislead the model and eventually, this may cause lower accuracy. For example, this is the case for Terasort, when increasing the sampling rate from 0.125 to 0.15 for the Uniform sampler. More points are chosen, but very few of them are picked in the regions where the execution time is high, thus the model's accuracy degrades. Such cases are avoided by utilizing Adaptive Sampling.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 3: R^2 and MAE for the benchmark applications

In conclusion, the provided models in cooperation with the sampling methods enable the system to create an accurate profile of the application even when the sampling rate is less than 10% of the points of the domain space. In terms of accuracy, we achieved R^2 values higher than 0.8 for all the benchmark applications, even when the sampling rate is lower that 10%. Similarly, the accuracy of the trained models increased rapidly in terms of MAE as well for increasing sampling rates. Finally, the profiling process is quite fast: Even when the Sampling Rate is 20%, the total time spent in training is not more than 1.5 seconds. The input space of our experiments consists of 135 discrete points for the Terasort case and 162 points for the SSSP and Pagerank cases. Sampling with 20% of these domains leads to 27 and 32 points respectively, thus the training time of our models is less than 1.5 seconds when there exist 32 points for training.

V. Related Work

Predicting the performance of applications running over virtualized resources concerning the workload is vividly researched in the literature. In [4], Kundu et al. proposed an iterative model training technique for Neural Networks with which the authors managed to predict the minimum possible Virtual Machine (concerning its resources) which would fulfill their objectives with respect to the SLAs. In an extension of this work, at [3], also utilized Support Vector Machines for the same objective. Their work achieved to highly accurate predictions, however the authors did not address the problem of sampling the input domain space, as we do in this work. Furthermore, Iqbal et al. in [16], propose a method with which, at first, identifies a workload pattern and secondarily builds a model capable to predict the application's capacity (the number of requests it can serve without violating given constraints). This work focuses on web applications and the prediction happens with regression models; *PANIC* on the other hand, provides a wealth of approximation techniques and the it also supports any application able to deployed over a cloud infrastructure.

Similarly, Do et al. in [6] presented a profiling technique which utilizes the Canonical Correlation Analysis, able to identify the relationship between the allocated resources and the application performance. This work targets to predict the performance of a newly allocated Virtual Machine when it is deployed in a specific host running other Virtual Machines. Our work differentiates from this, since our target is to provide an accurate application profile without having any knowledge about the provider. Other works focus on predicting specific application metrics based on I/O workload and access patterns such as [17], [18] and [19]. Our approach differentiates from them, as we propose a system where the user can define application level metrics which indicate the application performance.

VI. Conclusions

In this paper we addressed the problem of predicting the performance of a complex application deployed over virtualized resources. The goal of our work is to propose a system which obtains knowledge about the application by deploying it over a cloud infrastructure, in different deployment setups and then approximating its performance for all the possible setups. The application load, which is a key factor to the performance, is addressed in the same manner as the rest of the resources, contributing to a unified view of all the components that affect the application. The experimental evaluation indicated that such an approach can lead to an accurate prediction of the performance by actually deploying the application for only a very small portion of the deployment space. Furthermore, by utilizing a large number of approximation techniques, our system is able to quickly recognize the behavior of the application by picking the most suitable approximation model enabling the profiling process to terminate faster.

Acknowledgments

This work was supported by the European Commission in terms of the CELAR 317790 FP7 project (FP7-ICT-2011-8). Nikolaos Papailiou has received funding from IKY fellowships of excellence for postgraduate studies in Greece - SIEMENS program.

References

[1] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computingthe business perspective," *Decision Support Systems*, vol. 51, no. 1, pp. 176–189, 2011.

[2] J. Han, E. Haihong, G. Le, and J. Du, "Survey on nosql database," in *Pervasive computing and applications (ICPCA), 2011 6th international conference on*. IEEE, 2011, pp. 363–366.

[3] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling virtualized applications using machine learning techniques," *ACM SIGPLAN Notices*, vol. 47, no. 7, pp. 3–14, 2012.

[4] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao, "Application performance modeling in a virtualized environment," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010, pp. 1–10.

[5] A. A. Bankole and S. A. Ajila, "Predicting cloud resource provisioning using machine learning techniques," in *Electrical and Computer Engineering (CCECE), 2013 26th Annual IEEE Canadian Conference on*. IEEE, 2013, pp. 1–4.

[6] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 660–667.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[8] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*, 1987.

[9] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," DTIC Document, Tech. Rep., 1988.

[10] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[11] O. OMalley, "Terabyte sort on apache hadoop," *Yahoo, available online at: http://sortbenchmark. org/Yahoo-Hadoop. pdf,(May)*, pp. 1–3, 2008.

[12] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." 1999.

[13] S. Pettie, "Single-source shortest paths," *Encyclopedia of Algorithms*, pp. 847–849, 2008.

[14] R. G. D. Steel and J. H. Torrie, "Principles and procedures of statistics: with special reference to the biological sciences," 1960.

[15] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[16] W. Iqbal, M. N. Dailey, and D. Carrera, "Black-box approach to capacity identification for multi-tier applications hosted on virtualized platforms," in *Cloud and Service Computing (CSC), 2011 International Conference on*. IEEE, 2011, pp. 111–117.

[17] Q. Noorshams, D. Bruhn, S. Kounev, and R. Reussner, "Predictive performance modeling of virtualized storage systems using optimized statistical regression techniques," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013, pp. 283–294.

[18] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick, "Io performance prediction in consolidated virtualized environments," in *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 5. ACM, 2011, pp. 295–306.

[19] Kraft, Stephan and Casale, Giuliano and Krishnamurthy, Diwakar and Greer, Des and Kilpatrick, Peter, "Performance models of storage contention in cloud environments," *Software & Systems Modeling*, vol. 12, no. 4, pp. 681–704, 2013.