Evalix: Classification and Prediction of Job Resource Consumption on HPC Platforms

Joseph Emeras*, Sébastien Varrette†, Mateusz Guzek* and Pascal Bouvry†

*Interdisciplinary Centre for Security Reliability and Trust
†Computer Science and Communications (CSC) Research Unit
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg
`Firstname.Name@uni.lu`

Abstract. At the advent of a wished (or forced) convergence between High Performance Computing (HPC) platforms, stand-alone accelerators and virtualized resources from Cloud Computing (CC) systems, this article unveils the job prediction component of the Evalix project. This framework aims at an improved efficiency of the underlying Resource and Job Management System (RJMS) within heterogeneous HPC facilities by the automatic evaluation and characterization of the submitted workload. The objective is not only to better adapt the scheduled jobs to the available resource capabilities, but also to reduce the energy costs. For that purpose, we collected the resource consumption of all the jobs executed on a production cluster for a period of three months. Based on the analysis then on the classification of the jobs, we computed a resource consumption model. The objective is to train a set of predictors based on the aforementioned model, that will give the estimated CPU, memory and IO used by the jobs. The analysis of the resource consumption highlighted that different classes of jobs have different kinds of resource needs and the classification of the jobs enabled to characterize several application patterns of the users. We also discovered that several users whose resource usage on the cluster is considered as too low, are responsible for a loss of CPU time on the order of five years over the considered three month period. The predictors, trained from a supervised learning algorithm, were able to correctly classify a large set of data. We evaluated them with three performance indicators that gave an information retrieval rate of 71% to 89% and a probability of accurate prediction between 0.7 and 0.8. The results of this work will be particularly helpful for designing an optimal partitioning of the considered heterogeneous platform, taking into consideration the real application needs and thus leading to energy savings and performance improvements. Moreover, apart from the novelty of the contribution, the accurate classification scheme offers new insights of users behavior of interest for the design of future HPC platforms.

Keywords: RJMS, HPC, Classification, Machine Learning.

1 Introduction

Many organizations have departments and workgroups that benefit (or could benefit) from High Performance Computing (HPC) resources to analyze, model, and visualize the growing volumes of data they need to run their business. The size of the largest HPC platforms has dramatically evolved to attain hundreds of thousands of processors nowadays. Providing the energy and the cooling infrastructure to sustain such large systems is now more than ever a challenge. These tasks are becoming even more complex, as most of the current facilities comprise heterogeneous resources nowadays, either due to acquisitions at diverse period of time, or by the completion of the existing nodes with specialized hardware (GPU or CPU accelerators, FPGAs etc.) to achieve superior throughput for some specific workloads. Moreover, with the advent of the Cloud Computing (CC) paradigm and the widespread availability of virtualized computing resources, the classification and prediction of the most appropriate target for a given job is key to achieve a better efficiency and reduced energy costs. In this context, this paper presents the basic brick of the Evalix project, which aims at the automatic evaluation and characterization of HPC workload and user patterns to identify the jobs that may benefit from the underlying heterogeneity of the platform. For that purpose, we collected the resource consumption of all the jobs from a production HPC system operated within the University of Luxembourg (UL) on a period of three months. The analysis of these traces permitted to develop a model that link the profile of the submitted jobs with their actual usage pattern, whether in terms of CPU, memory or IO load. Another contribution of this article is the definition and implementation of a supervised machine-learning approach based on Support Vector Machines (SVMs) to characterize incoming jobs according to that model.

This paper is organized as follows: Section 2 reviews the context and motivations in the origin of the Evalix project. In particular, we demonstrate from the analysis of Resource and Job Management System (RJMS) logs within a production HPC platform the necessity to carry on a deeper characterization of users' jobs resource consumption. Then, the method used to collect and classify these usage patterns is presented in Section 3.1. A supervised learning approach is detailed in Section 4 to deduce the above classification for incoming jobs. A performance evaluation of the designed predictors is also provided. Section 5 reviews the related works. Finally, Section 6 concludes the paper and provides some future directions and perspectives opened by this study.

2 Context and Motivations

The performance of an HPC system is obviously determined by the unitary performance of the subsystems that compose it, but also by the efficiency of their interactions and management by the middleware. In these kind of systems, a central component called the RJMS is in charge of managing the users' jobs on the system's computing resources. The RJMS has a strategic position in the whole HPC software stack as it has a constant knowledge of both workload and resources. In order to improve the scheduling and resource management strategies, the workload of such systems has been widely studied and led to the construction of various models, as proposed by Lublin et al. in [1] and D. Fei-

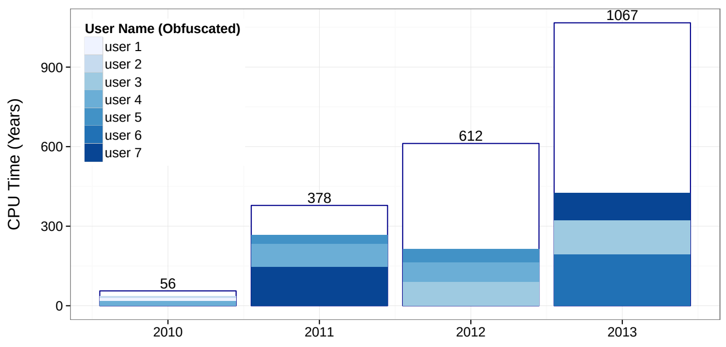


Fig. 1: Platform Yearly Utilization and Top Users Contributions.

telson in [2]. These models, were later used as a basis in scheduling techniques evaluations and optimization studies, such as [3] and [4]. However the aforementioned works were based on the study of the resources allocation to the jobs and not on their actual usage. We already discovered that allocation and usage often mismatch [5]. In order to have a more efficient scheduling of the jobs and a better resource management, it becomes necessary to get a better insight of the actual resource usage of the jobs. We have once again observed this divergence on a production HPC facility operated since 2007 by the UL. Users of this facility are people from three faculties and two Interdisciplinary centers within the UL, which cover various research topics such as bio-medicine, material science or security. An overview of the platform, its configuration together with its management is proposed in [6]. In this paper we study the usage of the *Gaia* cluster, the largest and most used of the UL HPC facilities. Composed of 151 nodes for a total of 2004 computing cores at the time of writing, it provides to more than 200 users a total computing power of 21.178 TFlops and relies upon the OAR [7] RJMS. During the last two years, *i.e.* between 2012 and end of 2013, almost three million jobs were launched, which gives an average throughput of one job every 23 seconds. Still in the same period, the average run time of jobs was about 33 minutes and many of them have a run time below the minute. The most frequent requested allocations are, by order of frequency: 1 core, 1 node, half a node (*i.e.* one socket) for single node allocations and 4 nodes, 2 nodes and 3 nodes for multi-node allocations. An average user have submitted more than 5,000 jobs to the RJMS and 9 users have more than 10,000 jobs.

As suggested in a previous study on user behavior [8], not all users have the same impact on an HPC infrastructure. In this work the authors classified users in three groups regarding their level of importance based on their utilization of the platform, assuming that top users were the most productive ones. This difference of importance between users on the UL HPC platform is well visible in Figure 1, which presents the yearly CPU time allocated to jobs. For each year, the 3 largest users in terms of job area (number of resources $\times$ run time) and their relative usage are presented. What is very remarkable is that since the launch of the platform the computational area of these large users is very

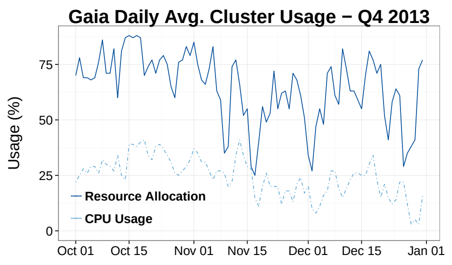


Fig. 2: Resource Allocation vs. actual CPU Usage. Over this time period 2 of the top users are responsible for 38% of total job area and a low CPU usage.

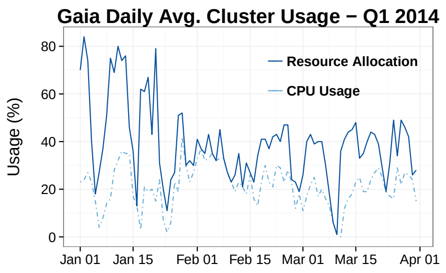


Fig. 3: Resource Allocation vs. actual CPU Usage. Time periods with different relative CPU usage patterns highlight the presence of different users.

large regarding the total year's area. For example in 2013, 3 users from the same laboratory are responsible for 41% of the whole platform utilization. This confirms the previous idea [8] of the existence of a class of a few top users who consume a large part of the computing power. In the study by Feitelson et al. [9] of nine logs taken from the Parallel Workload Archive (PWA)[10], it was found that in many cases the activity of a few individual users can dominate all other activity in the system. e.g. in the HPC2N log, one single user was responsible for 57.8% of the whole log activity and in the SDSC and LANL logs, the number of jobs produced by few users could be 5 to 10 times the average weekly total, but this only for a short period. However one might wonder if these really use the platforms in reasonable ways and if their jobs are efficient enough or if they spill some computational power. Indeed, the time consumed by them is the CPU time allocated to the jobs and not their actual CPU usage. In a previous work [5], we showed that the jobs' actual CPU usage was in many cases very different from the CPU allocation, even for jobs that are supposed to be CPU intensive.

In order to verify what is the actual CPU utilization, we reconstruct the CPU usage of the jobs in function of CPU allocation on the UL HPC platform. From RJMS logs, we compute the percentage of platform's CPU resources allocated to the jobs (taking into account failures and resource absence). To estimate the actual CPU usage on the computing nodes at this level, the data collected by the Ganglia monitoring system were used[1]. We measured the average system noise generated on the nodes by the Ganglia daemon as being negligible, i.e. less than 0.05% over one hour. Information retrieved this way is thus well representative of jobs CPU usage on the nodes. This information is retrieved at node level, so when several jobs share the same node we cannot say much about their respective CPU usage. However, this gives us a general tendency of the utilization vs. allocation. Figures 2 and 3 present the difference between CPU allocation by the RJMS and the actual CPU usage of the jobs. The darkest curve presents the percentage of CPU allocated by the RJMS regarding the total number of resources available

[1] Later on, another monitoring tool named *Colmet* [11] will be used.

in the system at this time. The lightest curve presents the aggregated CPU usage as reported by Ganglia. Thus the height difference between the two curves gives the relative CPU usage, which is frequently far from being 100% of what is allocated. Also, there exists periods with different CPU usage patterns. For instance in Figure 2, two users are responsible for 38% of job area, for a CPU usage of about 30%. These users belong to the top 3 users of the year 2013. If we take shorter time periods with different CPU usage patterns, for example in Figure 3 from *2014-01-01* to *2014-01-15* and *2014-02-01* to *2014-02-15*, we can observe the impact of the presence of particular users. In the first time period we have two peaks of high CPU allocation (around 80% of the full platform) with a CPU usage around 40%. During this period, two users are responsible for 45% of total job area. These are the same users that were predominant in Figure 2. On the contrary in the second time period the CPU allocation is lower (varying between 20% to 40%) for a high relative CPU usage. During this time period the two users account for 34% of total job area and a single user accounts for more than 43%. Not surprisingly these two are still the same ones as before and the user who has the largest job area is also one of the year 2013 top 3 users. This phenomenon is not isolated and other periods where these 3 top users are present have the same pattern (e.g. beginning vs. end of March 2014).
In the light of this observation, it seems that the presence of some particular users can have a strong impact on the platform usage pattern. Moreover it is also well visible that users have different resource consumption patterns, depending on the applications they run. This raises the following questions: *are there some typical user profiles that we could extract from the observation of the jobs?* and *could we benefit from this knowledge to predict users' future needs in terms of resources depending on the jobs they run?*
Answering these questions is central in the Evalix project, which aims at the automatic evaluation and characterization of HPC workload and user patterns to better adapt to heterogeneous resources. Detailing this project is clearly out of the scope of this paper. Here, we propose to look deeper into job's internals and measure what they use in terms of physical resources. Then based on the history of jobs consumption, a machine-learning approach is proposed to predict the expected resource consumption from incoming jobs upon submission. Obviously, the proposed supervised algorithm is the first step in the design of a framework able to cover Evalix goals. We now detail the analysis performed to classify the jobs consumption of the *Gaia* cluster over the considered 3-month period.

3 Job Consumptions Data Collection and Classification

3.1 Data Collection

Following the previous work on jobs consumption collection initiated in [5], we choose a monitoring of all the jobs processes, managed at the node level. In our previous work, we used the Linux */proc* virtual filesystem to gather information of resource consumption of all the processes that composed a job. Carefully

Platform	#cores	#users	Throughput (jobs/h)	Job Size Mode (cores)	Avg.Utilization (%)
ULHPC Gaia	2,004	84	32.3	12	45.4
Metacentrum	806	147	25	1	36.3
LLNL-uBGL	2,048	62	21.1	1024	56.1
PIK IPLEX	2,560	225	25.6	1	38
LLNL-Thunder	4,008	283	35.7	4	87.9
RICC	8,192	176	122	1	87.2
LLNL-Atlas	9,216	132	12.7	8	64.1
CEA Curie	93,312	722	82.2	1	29.3
ANL intrepid	163,840	236	12	2048	59.6
		Median	25.6	4	56.1
		MAD	15	4.4	26.8
		IQR	14.6	11	26.1
				General statistics	

Table 1: Comparison of the UL HPC Platform 3-month trace statistics with various other systems listed in the PWA [10].

polling the filesystem for collecting counters information at a one minute frequency enabled us to capture and analyze a 9 month trace on two production clusters at little cost. In this paper, we perform the monitoring of jobs consumption with a dedicated tool named *Colmet* [11], provided as a testing software by the OAR RJMS development team. Unlike Ganglia which is designed for monitoring a whole machine, Colmet is able to evaluate a set of processes. More precisely, it relies on Linux *taskstats* accounting feature coupled with the *cgroup* [12] kernel isolation mechanism to retrieve at low cost jobs consumption counters. Collected data is stored on a dedicated node in a file structured in Hierarchical Data Format v5 (HDF5). This storage type enables to process easily large volumes of data. The evaluation of this tool is not the topic in this work. However to give an insight, we measured the overhead induced by *Colmet* when monitoring resources by running some carefully selected benchmarks representative of HPC workloads (NAS Parallel Benchmarks (NPB) [13] version 3.3.1 for instance). Our evaluation demonstrate a performance overhead below 0.1% for the considered benchmarks, even with a frequency of 1s between each data collection step. This is by far the less intrusive tool we are aware of, for monitoring a given set of processes at a few seconds frequency and that provides such a large file storage capability. We collected a 3-month trace on *Gaia* cluster, from *2014-05-22* to *2014-08-19*. This trace is composed of 51859 jobs, belonging to 84 different users. The size of the trace stored in compressed HDF5 is about 10GB and contains 6.05×10^9 values from the different metrics retrieved by taskstats.

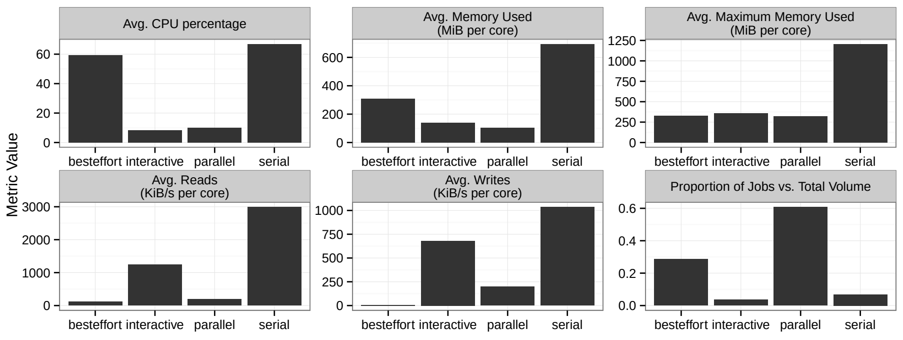
In order to compare *Gaia*'s workload with what is observable in other HPC sites we analyzed 8 of the most recent cluster logs of various size from the PWA. In Table 1, we present some of their characteristics along with our cluster's 3-month trace characteristics. We also provide the robust data dispersion indicators: Median Absolute Deviation (MAD) and Inter-Quartile Range (IQR), to estimate the differences between *Gaia* and these other systems. Despite its humble size when compared to large platforms such as CEA Curie or ANL Intrepid, we can see that *Gaia* still has a good job throughput and a relatively fair core utilization. Moreover its job mix is relatively close to what is visible in other systems and we can assume that our analysis and learning approach could also benefit to such other systems.

3.2 Analysis of Jobs Consumption

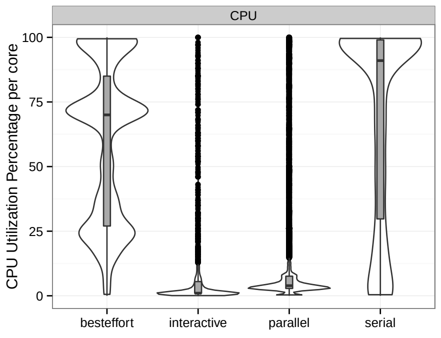
As *Colmet* collects temporal data, it would be possible to analyze job patterns during their execution. The analysis of temporal patterns in HPC applications is definitely worth doing, however due to the complexity of performing the learning with time series data, we will first focus on the average job consumption patterns. Thus, this work aims at the coarse grain analysis of jobs resource consumption and does not focus on temporal data for the moment. The analysis of the time series though is certainly a very interesting topic and shall be done in further works. To process and analyze this large amount of data, we first aggregate it on the job duration and allocated resources. This means that for each job we have its average CPU and memory usage, maximum memory reached, average disk IO reads and writes per second. The averages are given per allocated core. Thanks to the use of lightweight isolation mechanism,*Colmet* monitoring takes into account all the processes and threads that belong to a given job and thus ensures the completeness of job's usage data. In OAR RJMS, depending on the user application needs, a job can belong to one of the following classes:

– **besteffort** jobs are preemptible, low constrained multi-parametric jobs that are supposed to be CPU-intensive. A besteffort job will always be considered as being part of the besteffort class, regardless its number of allocated core.
– **interactive** jobs are for debugging purpose, they provide to the user a direct shell to his allocated machines.
– **serial** jobs are jobs requesting only one core.
– **parallel** jobs are traditional HPC jobs that request several cores.

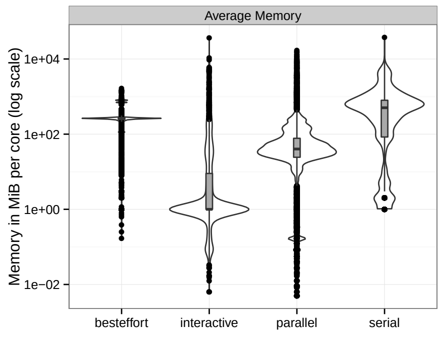
In order to visualize jobs that may be comparable, the Figure 4 exhibits jobs consumption statistics grouped by class of job. Figure 4a presents the jobs average consumption for each metric: CPU, memory (average and maximum) and IO (reads and writes) along with the proportion of jobs per class. If we focus on this last metric (bottom right figure), we can see that most of the jobs are parallel or besteffort. However, even though besteffort jobs account for 30% of total number of jobs, their area is much lower. In fact, they represent only 4.05% of the total job area while interactive jobs account for a relative area of 1.04% and the area of serial job is 5%. Thus *Gaia* workload is mainly composed of parallel jobs.
Figures 4b, 4c, 4d and 4e present violin plots, *i.e.* box plots with jobs probability densities. Violins are given per consumption metric and for each job class. The plot relative to the maximum memory usage is not shown as its data distribution is quite similar to the average memory usage. In Figure 4b, we observe that besteffort and serial jobs consume the most the CPU power. We also remark three distinct patterns in besteffort jobs corresponding to the largest job densities: around 25%, 75% and 100%. These patterns most probably come from different



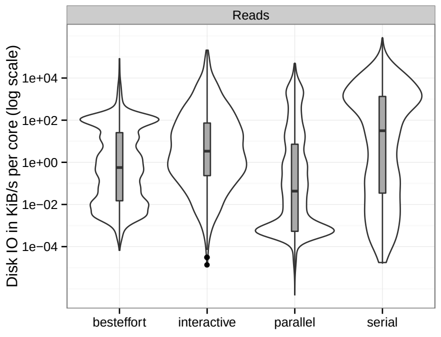
(a) Jobs Average Consumption per Class.



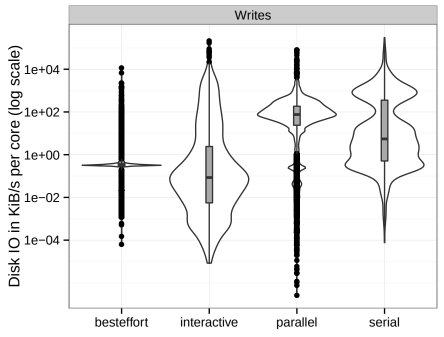
(b) CPU Usage Distribution & Density.



(c) Avg. Memory Usage Distribution & Density.



(d) Disk I/O Reads Distribution & Density.



(e) Disk I/O Writes Distribution and Density.

Fig. 4: Jobs Consumption analysis for the workload scheduled on the *Gaia* cluster between May 22th, 2014 and August 19th, 2014.

types of application. As we can assume, interactive jobs tend to have a very low CPU utilization. However what we were not suspecting is the low CPU utilization of parallel jobs. Almost all jobs have an average CPU utilization under 15%. Above that are only some outliers. In Figure 4c, we note that only

interactive jobs have a low memory usage. In this figure, the average memory usage is given per allocated core thus the global memory used by the job is depending on its size. We also observe that besteffort jobs have a very stable and predictable memory usage (on the order of 100MiB per core) while parallel and serial jobs have more variations. Finally from Figures 4d and 4e, we can spot that disk IO and in particular reads accesses have a very wide scale and spread distribution. We also witness that parallel jobs have the higher IO write values for median and density but lower IO read values. Thus parallel jobs tend to write more than others but read less.
Overall, the jobs that use the most the resources belong to the serial class. A very high proportion of these use around 100% of CPU but also a fairly high amount of memory (on the order of the GiB). What is quite surprising is the relatively low CPU utilization of parallel jobs. For a vast majority of the jobs, this value is around 5%. This is counterbalanced by a relatively high memory usage (on the order of the 100th of MiB per core) and for a lot of them, a high rate of disk writes. Moreover, this analysis exhibits the importance of the user factor: as little as 8 users among the 56 that ran parallel jobs during the considered period, account for an area of 80% of parallel jobs.

3.3 Classification of Jobs Consumption

As job consumption values distribution takes various forms from centered around the median to really sparse and with a wide range, we prefer here to adopt *first* a classification approach instead of using regression techniques to train the Evalix predictor. This preliminary step is expected to make the training easier with a better accuracy of the predictions. Each consumption metric will be divided into several classes of values and thus, the learning will be done on these classes and not on the continuous values. For the clustering of data, we chose to divide each consumption metric into four classes as depicted in Table 2.

The CPU consumption data was clustered linearly regarding its level CPU usage. For memory consumption, we applied a logarithmic clustering of the average and maximum memory usage; same for disk IO consumption (reads and writes) but given on average per second. Moreover, an expert knowledge based approach was chosen instead of automatic clustering algorithms since it was giving a stronger semantic meaning to classes. In particular, the logarithmic approach enables to compare the jobs based on their order of magnitude for memory usage or disk access, which is more suitable regarding the distribution of the resource usages. We compared our approach with three unsupervised learning classification methods: k-means, hierarchical trees clustering and gaussian

	CPU	**Memory**	**Disk IO**
Class 1	0 to 25%	up to 10MiB	up to 10KiB/s
Class 2	25 to 50%	from 10 to 100MiB	from 10 to 100KiB/s
Class 3	50 to 75%	from 100 to 1GiB	from 100 to 1MiB/s
Class 4	75 to 100%	above 1GiB	above 1MiB/s

Table 2: Clustering classes for consumption metrics.

User Average Consumption	**# of Users**
CPU Intensive	11
CPU and Memory Intensive	7
CPU and IO Intensive	1
Memory Intensive	15
Memory and IO Intensive	3
IO Intensive	4
All Resources Intensive	1
Low Resource Usage	10
Medium Usage – Unclassified Users	54

Table 3: User Classification based on Average Resource Usage over the Jobs Monitoring Period (84 active users) excl. interactive jobs.

mixtures. None of them was satisfying enough compared to our naive manual clustering approach: *Hierarchical trees clustering* gives although too much (over 10) or too few classes (only 2), depending on the level of intra-class heterogeneity chosen. *Gaussian mixtures* led to similar problem and when the number of classes is limited to a reasonable number (between 4 to 8), a few very dense classes are produced while others are very sparse. Finally, the *k-means clustering* also reproduces the same problem with very sparse classes.
To get back at our first initial question: *are there some typical user profiles that we could extract from the observation of the jobs?* It is now easy to detect some classes of users that have always the same usage pattern. We define that, if on average a user belongs to classes 3 or 4 for a given consumption metric, he uses intensively the resource. However, if the maximum class attained in all this user's jobs is 1 or 2, we define his use of the resource as low. Thus a user highly relying on several resources will be marked as so and a user consuming lowly all the resources will be categorized as having a low resource usage. We have to be careful not to take into account interactive jobs, these being debugging or setup jobs, they do not reflect an application behavior. Applying this methodology on the jobs consumption classes gives us the user classification presented in Table 3. What is satisfying is that among 84 active users, 30 of them have a stable behavior and can be classified very simply. It is also interesting that a total of 10 users were identified as lowly using all the resources. The area of their jobs only accounts for 3.12% of the total job area but with an average CPU usage of 18% which cause a loss in CPU time of more than 5 years. As for the 54 users that could not be classified with the above methodology, either they have a stable behavior but their jobs are mixed between different resources consumptions or they run applications with different resource usage patterns. For instance, the top 3 users on the year 2013 fall into this "*Unclassified*" category yet we are now able to better understand their profile. Concerning the first two users who had an overall low CPU usage, one of them has most of his jobs being composed of a medium to low consumption on all the metrics and few jobs with a high CPU usage. On the data observed from this user's jobs we could not see any correlation between the CPU usage and memory or disk usage. What is the most

probable is that his jobs are not belonging to one particular class but correspond to a mixed usage patterns. The second user seems to have two kinds of jobs, one with a very high CPU usage and a moderately high memory usage, and another type, which represent most of his workload, shows a very low CPU and memory usage pattern. In all his jobs, the volume of disk reads and writes is very small. What is very interesting with this user's jobs profile is that their memory usage is directly correlated with their CPU usage. We can be quite confident that we are in presence of a user whose workload is composed of two distinct types of jobs. For the third user who was showing a high CPU usage on the Figure 3, there are probably more than two job classes. For the CPU usage, 60% of his jobs belong to class 1 and 32% of his jobs are of class 4, but despite the CPU class we also witness different memory usage patterns (with low and high usage) that are not correlated with the CPU usage. However, what is well visible for this user is the temporal correlation of the consumption. Indeed, this user's jobs that are submitted within a short time frame tend to have a very similar CPU or memory usage. This is a very interesting property that could be useful in an online classification. More generally, we need a more advanced mechanism to assort the *Unclassified* jobs consumption, in particular to take into account not only the user name but also all the job input parameters. That's the object of the machine-learning approach proposed in the next section, with the idea that by perusing the history, we will be able to predict all future job consumption classes based on user query to the RJMS.

4 A Supervised Learning Algorithm for the Complete Prediction of Job Resource Consumption

Several supervised learning techniques coexist in the literature. Among the most used ones are Neural Networks and SVM algorithm [14]. Although they differ in their mechanism, they can both be used for data classification and regression analysis. In [15], the comparison between both techniques revealed that SVM is generally more efficient. More precisely, at the price of a higher computation time, SVM is able to compute models that generate predictions with a lower error rate. As the training of our models is done offline from a trace extraction, models computing time is not a constraint and we prefer the approach that gives the best results. To perform the supervised learning and train the SVM-based predictors on jobs consumption data, we choose the reference implementation proposed in the *libsvm* [16] library (version 2.6).

4.1 Metrics for Machine Learning Performance Assessment

In our previous classification of jobs consumption, a classification of data within four classes has been chosen to have a finer evaluation of the consumption. Indeed, a two-class classification seemed too restrictive as it would only tell if a given resource usage was either *high* or *low*. With four classes we have the possibility to express more precisely at which level is the resource consumption.

Nevertheless, SVM is originally designed for binary classification problems. In consequence, our considered multiclass classification problem needs to be transformed to a form of binary classification. Generally this is done either with a *one-against-one* or a *one-against-all* voting scheme. The first method decomposes the original problem into several two-class classification problems. The second one treats each class separately and data either belongs to the class or not the class (i.e. any of the other classes). In a study comparing multiclass SVM problems for machine learning [17], it was shown that among several voting scheme, the *one-against-one* technique is commonly seen as the most suitable. Consequently, this approach has been considered for Evalix predictors. Also, the usual classification performance indicators are not useful. For example sensitivity, specificity and likelihood are meant to be calculated for two classes only. In [18], the authors proposed three performance indicators that can be used for the evaluation of multiclass classifiers:

1. **Accuracy**, which gives the proportion of observations that were correctly classified. Derived from the confusion matrix, the multiclass accuracy is the average of the accuracies obtained from each class. This metric provides the **information retrieval rate** performed by the learning.
2. **Area Under the ROC Curve (AUC)** which comes from the radiologic community to judge the discrimination ability of statistical methods. The AUC measures the **probability to correctly classify a random sample**.
3. **Cohen's kappa** measure of agreement. This indicator aims to compensate for classifications that may be due to chance. In [19], the use of Kappa is proposed as a standard meter for measuring the accuracy of all multi-valued classification problems. A kappa value over 40% is generally considered to be a moderate agreement and over 60% a good agreement.

In the context of Evalix predictors, the above three metrics were considered as *complementary* performance measures. On the one hand, while the accuracy remains the most widely used indicator due to its simplicity, it was showed in [20] that this metric alone can be misleading under skewed class distribution, which is the case for some data in the present study. On the other hand, the kappa metric suffers from several undesirable effects: first, kappa may be low even though there are high levels of agreement and that individual ratings are accurate [21]. Then, and that's more problematic in our case, its value is influenced by data distribution and as a result, kappa values should not be compared across studies [22]. As regards the AUC evaluation, we use a generalized pairwise comparison approach as proposed in [23]. Given the inherent advantages of this metric [24] (better standard error as the number of test samples increases etc.), AUC will remain our most important evaluation criterion with precedence over the accuracy and kappa indicators.

4.2 Training and Evaluating the Models

Input Data Selection. Mandatory information to submit a job in OAR RJMS is: user name, submission queue, number of resources asked, maximum time requested (or walltime), type of job (interactive or batch). Using the information on OAR configuration on the cluster, we are able to determine also the class of the job (besteffort, interactive, parallel or serial) so we can add this information in the training. Iteratively we tested the prediction results in function of the job characteristics used as input in the learning. The best results obtained, which are presented later in this article, used as learning input the **user name**, the job **submission queue**, the **number of resources reserved**, the job **walltime**, the **job type**, whether the job was an **advance reservation** or not and the **job class**. For instance, the job name (which is optional in OAR) was of no interest for the training. Actually we got worse results with the trainings that included this information than the ones that did not. The explanation is quite simple: 26% of the jobs have no name, and for those who have, these names reflect for many user either the version of the code run or the application input parameters. This means that many jobs have different names that slightly differ but actually correspond to the same job with different input parameters. In consequence, this information disrupts the learning process. Based on the information given by the user at submission time, we train one predictor per consumption metric. This means that from the history of the input parameters of each job, associated with their resource consumption, we compute the support vectors and the models that will describe this relationship. Thus we will have five consumption models: CPU utilization, average memory, maximum memory, disk IO reads and writes; each computed from jobs input parameters.

Finding the Best Training Parameters. The performance of SVM is very dependent on the choice of parameters [25]. To ensure a good learning of the consumption data we evaluated two of the most used kernels: polynomial and RBF (radial), and for each of them using a grid search to determine the best hyper-parameter set. For both kernels we evaluated the error rate and dispersion with a gamma between $10^{(-3:3)}$ and a cost between 1 to 5. For polynomial kernel we tested a degree between 1 to 3. With the best parameter sets for each kernel and for each model, we found that the polynomial kernel performed better than RBF during the prediction evaluation (slightly better Accuracy, but 1 to 2% better AUC and Kappa). Thus we present only the results for the polynomial kernel. The best parameter set for our classification problem was thus a degree 3 polynomial kernel, with a gamma value of 1 and a cost of 2. This set gave us an error rate of 0.179 and a dispersion of 0.007. Since our data sample is relatively large (over 50,000 samples), a test set of 10% of data size and a cross-validation technique at training phase is recommended [26] to ensure a low variance of the results. Thus for the training of our models we first extract 10% of data that will be used as test data. Then, using the best parameters we train our models on the remaining 90% using a 10-fold cross validation with a validation set of 10% at each fold, then we also compute the predictive accuracy of the models on their respective test sets. This gives the results presented in Figure 5 and Table 4. The 10-fold cross validation ensures statistically stronger results than

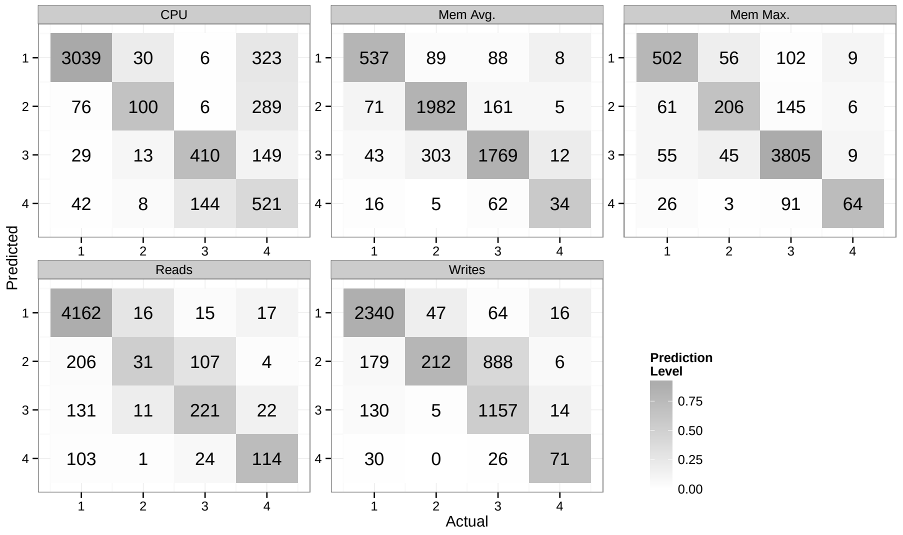


Fig. 5: Test Data Confusion Matrices.

the test evaluation itself as the process of selecting a test data set and comparing it with a train data set is done 10 times on disjoint sets.

Models Evaluation. Figure 5 presents the confusion matrices of the classification of test data for each of the five consumption models. On the x-axis are the actual classes values. On the y-axis are the predictions. The diagonal in the confusion matrices represent the jobs that were correctly classified by the predictor while off-diagonal elements are those that are misclassified by the predictor. The higher the diagonal values of the confusion matrix the better it is, since it indicates many correct predictions. The prediction level presents in a scale of grey, for a given class the rate of job classifications. What we observe is that generally the predictions are accurate: the darker cells of the matrix (corresponding to the higher classification rates) are located on the diagonal. However, we can also observe that even though the successful classification rate is high, the reverse is not always true. This means that for a given predicted class, the proportion of jobs that actually were belonging to this class can be low. For example, in the Disk Writes model, over all the jobs predicted as class 2, 179 were belonging to class 1, 212 to class 2, 888 to class 3 and 6 to class 4. In this example, only 16.5% of the jobs that were predicted as class 2 were actually belonging to class 2.
In order to perform the evaluation of the models taking into account this phenomenon, instead of simply looking at the confusion matrices, we also use the earlier defined performance indicators, with the results depicted in Table 4. It shows the model accuracies of the 10-fold cross validations along with test data evaluation. Even though the 10-fold cross validation is statistically stronger, the evaluation on test data is also interesting as it enables to compute not only the model accuracies but also the other performance indicators AUC and kappa.

		Test Data		
Model	**10-fold Accuracy**	**Accuracy**	**AUC**	**kappa**
CPU	79.4%	78.5%	75.6%	60.8%
Mem.Avg.	83.4%	87.3%	81.1%	73.2%
Mem.Max.	88.3%	93.8%	78.4%	68.8%
Writes	71.3%	72.9%	78.1%	57.5%
Reads	87.0%	87.3%	70.6%	53.4%

Table 4: Accuracy Values of Trained Models and Test Data Prediction Scores.

The cross validation shows a good accuracy for the five models, around 80 to 88% of the jobs are correctly classified. The best accuracies are for the Memory and disk Reads models, while the CPU and disk Writes show a slightly worse accuracy. The accuracies computed from test data are, for all the models except CPU, seemingly higher than the ones obtained from the cross validation, this illustrates why the cross validation is very important for a stronger evaluation of the model. Test data AUC values are quite good, the probability to correctly classify a sample from this set is around 80% for Memory and Disk Writes, 75% for CPU and 70% for disk Reads. It could be counter-intuitive that disk Reads has the lowest probability even though it has one of the highest accuracies. In fact this comes from many jobs from the test data set belong to class 1 and were actually classified as class 1, this gives a very good diagnostic rate for the class 1. However this is not necessarily the case for the other classes, in particular class 2 and class 3. This is an example of the influence of data distribution skewness on the accuracy value. It is the same phenomenon for CPU that also shows a high density of jobs in class 1 correctly classified but class 4 shows a less good diagnostic rate. This perfectly highlights the interest of AUC over the sole accuracy as a performance indicator. For the kappa indicator, the best performance is obtained from the CPU and Memory models. Performance indicators for the different models evaluated on test data showed quite good results. Accuracy, AUC and kappa were giving fair to good scores for each model and the 10-fold cross validation accuracies were still remaining good. As the statistical properties of the cross validation ensures a more reliable accuracy evaluation, we will compute the others indicators with the same method from the full data set.

4.3 Evaluating the Predictions on the Full Data Set

To evaluate deeper the performance of the models computed in the previous section and to ensure a stronger statistical result, we split the initial data set into 10 extracts of 10% of its size.

Each of these extracts will be used as different test sets to evaluate the prediction performance. By this means we will mimic the 10-fold cross validation process that was used in the predictor training phase. Thus the performance evaluation of the predictions is not computed from a test set extract as was done for the training validation, but on different subsets that cover the full data. For each test set and for each metric we compute the Accuracy, AUC and kappa.

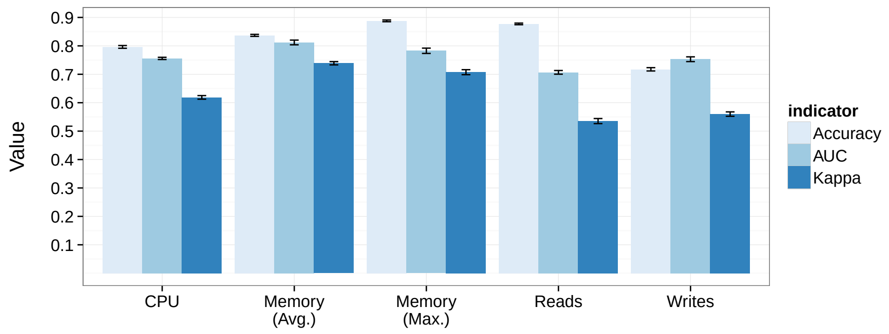


Fig. 6: Model Prediction Evaluation Scores.

The result of the performance indicator averages along with the 95% confidence intervals obtained are presented in Figure 6. On average the model accuracies are very good: either close to or above 80%, except for Disk Writes model that shows a lower accuracy of 71%. This corresponds to an information retrieval of 71% to 89%, computed solely from the information given at the submission time of the job. For CPU and Memory models the Cohen's kappa indicator is above 60% which is considered as a good value. For Disk Reads and Writes, the value is lower than for the others but is still over 40% which is considered as an acceptable kappa value. In the same time for each of the five models, the AUC indicator shows a fair to good probability to correctly predict the resource consumption class of the jobs. We remind that, for multiclass problems the AUC shows interesting statistical properties and lower dispersion regarding other indicators. This holds particularly when dealing with large samples, as is our case. This is why we chose this indicator as our main evaluation criterion.
For the CPU, Memory and disk Writes models the multiclass AUC is larger than 75% which corresponds to a good prediction score. For the disk Reads model the AUC is 70% which is still a fair value. Considering the fact that we are dealing with a multiclass problem and that we have four possible classes, a random prediction would give a probability of 0.25 for each class. Thus, a probability of having an accurate prediction being between 0.7 and 0.8 is a good score. We evaluated our five multiclass models with three different criteria: the accuracy which gives the amount of information correctly predicted, Cohen's kappa that compensates for predictions made by chance and the AUC which gives the probability to correctly classify a random sample. The three performance indicators gave us a good overall performance of our consumption models. The Disk Reads model in spite of a high accuracy is the one that shows the lowest prediction score considering the AUC (71%) and kappa (54%) values but is still considered as having a fair prediction score.Based on these observation we can say that the information provided by the user at job submission time is already a reliable source of information to predict the resource consumption of a job, which raises many scheduling optimization possibilities.

5 Related Work

The idea of using machine learning techniques, regressions and history analysis to predict the jobs characteristics based on their input parameters as already been addressed in several works.

In [27], two bioinformatics applications were used as benchmarks for an empirical assessment of the suitability of several machine learning techniques for the prediction of the size and time used by the jobs. In [28], Tsafrir et al. used predictions of job runtimes to correct user estimations. They showed that using the user estimated runtime of the job as a kill-time and using their proposed predictions as a runtime estimate, enabled to propose a simple scheduler respecting FCFS and EASY properties but with significant improvements in performance, predictability and accuracy. The generated prediction of job runtime was simply an average of the runtimes of the last two jobs by the same user. They also highlighted the importance of using recent data for predictions rather than a long history. In a similar work by Smith et al. [29], four workloads from the PWA were used to study the characteristics of the jobs resource requirements to the RJMS to predict application runtimes. The jobs characteristics used for the predictions were the job type (batch or interactive), submission queue, user name, number of nodes requested, among others, then used a genetic algorithm to find job templates used for the predictions. This method resulted in prediction errors of 40 to 60 percent of mean run times on the four workload studied which was a good improvement (14 to 60% lower error) regarding previous works [30] where classifying jobs characteristics used for the predictions were user name, parallelism level and submission queue. The approach used by Smith et al. using more criteria for the predictions was more efficient.
However, in spite of a wide literature on job runtimes prediction based on the jobs characteristics at submission time, very few job address the study of the prediction of jobs resource consumption. In [31], the authors used four application benchmarks, each corresponding to a particular resource consumption pattern: CPU, Memory, IO, Network. With the profiling of these applications, using the Ganglia monitoring system and ran within a Virtual Machine (VM), they trained a 3-Nearest Neighbor classifier. This classifier is to be used later to categorize a given application into CPU-bound, IO-bound, Network-bound or Idle. Based on this, around ten real world applications and benchmarks are classified and this classification is used as an input to an ad-hoc scheduler on a small cluster of workstation. They showed that the knowledge of the applications resource requirements enabled the scheduler to perform a better system throughput of about 22% by not scheduling applications with the same consumption pattern on the same nodes. Our work differs from this as we did not use a set of few benchmarks for the training of the predictor but we analyzed a real 3-month trace of our own users' job consumption. To the best of our knowledge, the only analysis of the jobs resource consumption on a large trace from a production HPC cluster was from our previous work [5]. By using the analysis of the workload coming from our own facility instead of using benchmarks, we remove the bias of training with applications unadapted to our users' workloads. We also

guarantee that data used for classification and prediction is realistic while more accurate predictions of the future jobs consumption are ensured.

6 Conclusion

In this work and in the context of the Evalix project, we collected, analyzed and classified a trace of all the jobs resource consumptions during a period of three months on a production HPC cluster. The outcome of this work is threefold. First, the analysis of the resource usage by the jobs depending on their class enabled to differentiate different kind of resource needs. Interactive jobs have a very low CPU and memory usage but can have high IO usage, besteffort jobs are very CPU intensive with few IO and medium memory usage and serial jobs are using the resources quite intensively.
Then, the classification of the jobs regarding the CPU, memory and disk IO consumption enabled to characterize the activity of 20 of the users active at that period (24% of the total number), whose applications are always of the same type. We classified these users as being intensive in one or several of these metrics: CPU, memory, disk IO. We also identified 10 users (or 12% of the total number) that always use very few resources. The CPU time loss caused by the activity of these accounts is in terms of years.
Last, we build a consumption model based on the classification and trained a set of predictors based on the jobs input parameters. We validated our prediction models with the comparison of three criteria and showed that the information retrieval rate is between 71% and 89%, and the probability of predicting the correct class is quite high: from 0.7 to 0.8.
These results lead to several optimizations of the RJMS and the scheduling. Firstly, the most immediate optimization is to benefit from the knowledge of the class of the job to make a scheduling aware of jobs resource needs, as proposed in [31]. The principle was simply to not schedule applications with the same consumption pattern on the same nodes and this enabled to perform a better system throughput. By not scheduling several serial jobs on the same resources, or by mixing on the same resources interactive and besteffort jobs (which show complementary patterns), this will positively impact the jobs. Moreover, the preliminary classification of the jobs is done based on an expert knowledge approach, however it will be interesting as a future direction to evaluate fuzzy logic algorithms to better handle uncertainties that come from the data dynamics.
Secondly, the prediction of the consumptions at job submission time will enable to refine the scheduling aware of jobs resource needs. With the prediction, the scheduling will be able to load balance the jobs on the heterogenous resources to obtain a better performance and energy efficiency. The predictor needs to be updated frequently and integrate a two-level prediction. First compute a prediction from a consumption model derived from the recent activity on the cluster, i.e. from the last months or weeks. Then for jobs that cannot be predicted by this means, to use a larger model containing an annual history of jobs consumptions.
Finally, with the analysis of the job resource consumptions, one can compute the

real cost of the jobs of the users on the platform. Moreover, most recent schedulers embedded in the RJMS use fair-sharing strategies to avoid the monopoly of the resources by the largest users or to grant more resource hours or higher priorities to a certain group of users. This real usage cost must be balanced by the resource request provided by the user at submission time. This would give a usage information that, integrated into the fairness score, will enable the user to pay the right price for his computation and to get a feedback of his behavior on the platform.
With the ability to automatically evaluate and characterize HPC workload and user patterns, and with a model of the cost of jobs, the Evalix framework will provide a highly efficient usage in terms of resources and infrastructure costs, along with a better adaptation of the jobs to heterogeneous resources. More precisely, our study offers new insights to guide the future partitioning of the computing platform: it is now possible to define in an accurate manner several sets of computing resources that will fit the analyzed heterogeneous usage patterns. Coupled with a wrapper at the RJMS level that schedules the incoming jobs according to their corresponding partition, substantial gains can be obtained, whether at the level of the computing of energy efficiency. Our first experiments based on a naive trace replay simulator reveal a potential energy efficiency improvement between 5 to 10%. Part of our short-term perspective for this work consists in consolidating these results and integrating the proposed classification/prediction scheme within the RJMS of our HPC platform to evaluate experimentally its effectiveness. Another mid-term objective is related to the extension of our work to a more accurate temporal analysis of the collected traces, to better take into account the user pattern changes over sliding period of time.

Acknowledgments: The experiments presented in this paper were carried out using the HPC facility of the University of Luxembourg. Many thanks are also due to all those who participated in collecting and distributing the logs available through the PWA and used in Table 1.

References

1. Lublin, U., Feitelson, D.: The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. Journal of Parallel and Distributed Computing (2001)
2. Feitelson, D.: Workload modeling for performance evaluation. In: Performance Evaluation of Complex Systems. Volume 2459 of LNCS. (2002) 114–141
3. Feitelson, D., Jettee, M.: Improved utilization and responsiveness with gang scheduling. In Feitelson, D., Rudolph, L., eds.: JSSPP. Volume 1291 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1997) 238–261
4. Cao, J., Zimmermann, F.: Queue scheduling and advance reservations with cosy. In: Parallel and Distributed Processing Symposium. (2004) 63
5. Emeras, J., Ruiz, C., Vincent, J.M., Richard, O.: Analysis of the jobs resource utilization on a production system. In: JSSPP. LNCS. Springer (2013)
6. Varrette, S., Bouvry, P., Cartiaux, H., Georgatos, F.: Management of an Academic HPC Cluster: The UL Experience. In: Proc. of the 2014 HPCS conference. (2014)

7. Capit, N., Costa, G.D., Georgiou, Y., et al. A batch scheduler with high level components. CCGrid (2005) 776–783
8. Wolter, N., McCracken, M.O., Snavely, A., et al. What's working in hpc: Investigating hpc user behavior and productivity. CTWatch Quarterly **2** (2006)
9. Feitelson, D.G., Tsafrir, D., Krakov, D.: Experience with using the parallel workloads archive. Journal of Parallel and Distributed Computing **74**(10) (2014)
10. Feitelson, D.: Parallel workload archive
11. : Colmet. [online] `https://github.com/oar-team/colmet`
12. : Linux Kernel. [online] `https://www.kernel.org/` — Taskstats, Cgroups
13. Bailey, D.H.: Nas parallel benchmarks. Springer (2011)
14. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**(3) (1995)
15. Duan, R., Nadeem, F., Wang, J., Zhang, Y., Prodan, R., Fahringer, T.: A hybrid intelligent method for performance modeling and prediction of workflow activities in grids. In: Proc. of the 2009 CCGRID conference. (2009) 339–347
16. Chang, C.C., Lin, C.J.: Libsvm: A library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3) (May 2011) 27:1–27:27
17. Hsu, C.W., Lin, C.J.: A comparison of methods for multiclass support vector machines. Neural Networks, IEEE Transactions on **13**(2) (Mar 2002) 415–425
18. Szollosi, D., Denes, D.L., Firtha, F., Kovacs, Z., Fekete, A.: Comparison of six multiclass classifiers by the use of different classification performance indicators. Journal of Chemometrics **26**(3-4) (2012) 76–84
19. Ben-David, A.: Comparison of classification accuracy using cohen's weighted kappa. Expert Systems with Applications **34**(2) (2008) 825 – 832
20. Provost, F.J., Fawcett, T., et al.: Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In: KDD. Volume 97. (1997) 43–48
21. Uebersax, J.S.: A generalized kappa coefficient. Educational and Psychological Measurement **42**(1) (1982) 181–183
22. Feinstein, A.R., Cicchetti, D.V.: High agreement but low kappa: I. the problems of two paradoxes. Journal of Clinical Epidemiology **43**(6) (1990) 543 – 549
23. Hand, D., Till, R.: A simple generalisation of the area under the roc curve for multiple class classification problems. Machine Learning **45**(2) (2001) 171–186
24. Bradley, A.P.: The use of the area under the {ROC} curve in the evaluation of machine learning algorithms. Pattern Recognition **30**(7) (1997) 1145 – 1159
25. Duan, K., Keerthi, S., Poo, A.N.: Evaluation of simple performance measures for tuning {SVM} hyperparameters. Neurocomputing **51**(0) (2003) 41 – 59
26. Guyon, I.: A scaling law for the validation-set training-set size ratio. AT&T Bell Laboratories (1997)
27. Matsunaga, A., Fortes, J.A.B.: On the use of machine learning to predict the time and resources consumed by applications. In: CCGrid. (2010)
28. Tsafrir, D., Etsion, Y., Feitelson, D.: Backfilling using system-generated predictions rather than user runtime estimates. Parallel and Distributed Systems, IEEE Transactions on **18**(6) (June 2007) 789–803
29. Smith, W., Foster, I., Taylor, V.: Predicting application run times using historical information. In Feitelson, D., Rudolph, L., eds.: JSSPP. Volume 1459 of LNCS. Springer (1998) 122–142
30. Gibbons, R.: A historical application profiler for use by parallel schedulers. In: JSSPP. Volume 1291 of LNCS. Springer (1997)
31. Zhang, J., Figueiredo, R.: Application classification through monitoring and learning of resource consumption patterns. In: IPDPS. (April 2006)