Job Placement Advisor Using Turnaround Predictions for HPC Hybrid Clouds

Renato L. F. Cunha, Eduardo R. Rodrigues, Leonardo P. Tizzei, Marco A. S. Netto
IBM Research, Sao Paulo, Brazil

***Abstract*—Vast literature exists on performance impact and cost benefits of using cloud for High Performance Computing (HPC) applications. As cloud becomes mature to execute HPC applications, companies and research institutions are starting to setup hybrid environments, mixing on-premise clusters and cloud infrastructure. From the end-users' perspective a major challenge is to decide where a job should be placed considering execution time and queue wait time to access the on-premise cluster. Several techniques are available to obtain turnaround predictions and it is well known they tend to be far from accurate. This paper investigates how to use execution and queue wait time predictions to make job placement decisions in hybrid HPC cloud environments and the impact of their inaccurate estimations on jobs' turnaround times. We used job traces from real supercomputing centers to run our experiments, and compared the performance between environments using real speedup curves. Our main finding is that depending on workload characteristics, there is a turning point where predictions should be disregarded in favor of a more conservative decision to minimize job turnaround times.**

***Index Terms*—HPC; Cloud Computing; HPC Cloud; Hybrid Cloud; Advisory System; Cloud bursting**

I. Introduction

Cloud computing has become an essential platform for several applications and services, including those with High Performance Computing (HPC) requirements. A clear trend is the use of hybrid clouds comprising on-premise[1] and remote resources. During peaks of demand, jobs are submitted to the cloud rather than being submitted to on-premise clusters, which can have long queue waiting times compared to cloud resource provisioning times.

Current work on HPC cloud has mainly focused on understanding the cost-benefits of using cloud over on-premise clusters [1–7]. In addition, the aim has been to evaluate the performance gap between cloud and on-premise resources [8–11]. Even though cloud typically has slower internal network speeds than on-premise resources, bursting jobs to the cloud can still provide better overall performance in overloaded environments. However, users still have to decide where to run their jobs at any given moment. Supporting users in such decision is the objective of this work.

The decision of keeping a job on-premises or moving to the cloud depends on several factors, but we focus on three of them: job queue waiting time, execution time and the relative performance of the cloud compared to the performance of on-premises machines. If users knew how long their job would wait in the local queue and how long it would take to run in both environments, then they could obtain the optimal turnaround time. However, this information is not known in advance.

Our strategy is to estimate the waiting time and execution time using historical data. The estimation method is based on the work of Smith [12], and is used within our decision support tool, the *advisor*, to predict job runtimes and wait times in on-premise clusters. Based on a cloud *versus* on-premise performance ratio, this tool computes a turnaround time estimate for both environments. In addition, we also compute a measure of the advisor's uncertainty that is compared to a threshold. Whenever the uncertainty is below a threshold computed by a cutoff function, the user is advised to run in the environment with the shortest turnaround time.

We evaluated the advisor with traces of real job queues and show its benefits under different performance ratios. The advisor processes queue historical logs to extract prediction labels, waiting and execution times, and features, which can be either fields from the original logs, such as submission time, requested time and requested number of processors, or they can be derived from the queue state, e.g. queue size, processor occupancy and queued work. The prediction is based on an Instance-Based Learning (IBL) algorithm [13] that relates a new incoming job with similar jobs in the history. The predicted waiting time or execution time is computed by a function of the labels of similar jobs. These predictions are then combined in the advisor to make allocation decisions. We evaluated the advisor's decisions using the "saved-time" metric and a conclusion of our evaluation is that improving scalability in the cloud is critical, not only to improve overall performance, but also to minimize penalties of mistaken placement decisions.

In summary, our main contributions are:

- Decision support tool based on runtime and wait time predictions for hybrid HPC cloud environments (§ IV);
- Cutoff function to make conservative resource allocation decisions that considers the uncertainty of the runtime and wait time predictions (§ IV, § V);
- Evaluation of the advisor using traces from real supercomputing workloads with lessons learned on the management of prediction uncertainty (§ V); we also validated an existing technique for runtime and wait time predictions;
- Evaluation of the advisor using real speedup curves from applications executed in six different environments: three on-premises and three cloud-based (§ V).

[1]In this work we use the terms "local" and "on-premises" interchangeably.

II. Related Work

Research efforts related to our work are in three major areas: metascheduling, job waiting time prediction, and runtime prediction. Solutions from these areas come from cluster and grid computing but can be applied to HPC cloud.

A. Metascheduling

Literature on metascheduling is extensive, mainly from Grid computing and more recently on hybrid clouds and inter-cloud environments. For instance, De Assunção et al. [14] evaluated a set of scheduling strategies to offload jobs from on-premise clusters to the cloud. These strategies consider various backfilling approaches that check the current status of job waiting queue; decisions are made when jobs arrive and complete execution. Sabin et al. [15] study metascheduling over a heterogeneous multi-site environment. Scheduling decisions rely on expected completion time in each site; however errors on estimations of expected completion time are not consider in these decisions. Sotiriadis et al. [16] introduced a state-of-the-art review on metascheduling related technologies motived by inter-cloud settings. The same research group presented a study on the role of meta-schedulers for inter-cloud interoperability [17]. Garg et al. [18] introduced two heuristics for scheduling parallel applications in grids considering time and cost constraints. Sabin et al. [15] studied scheduling of parallel jobs on heterogeneous multi-site environments. Malawski et al. [19] studied task planning over multiple cloud platforms using a mixed integer nonlinear programming having cost as their optimization goal.

B. Queue time predictions

Measuring how long a job will wait in a queue before its execution is a key component for deciding where jobs should be placed. There are several techniques available in literature. For example, Li et al. [20] investigated methods and algorithms to improve queue wait time predictions. Their work assumes that similar jobs under similar resource states have similar waiting times as long as the scheduling policy and its configuration remains unchanged for a considerable amount of time.

Nurmi et al. [21] introduced an on-line method/system, known as QBETS, for predicting batch-queue delay. Their main motivation is that job wait times have variations that make it difficult for end-users to plan themselves and be productive. The method consists of three components: a percentile estimator, a change-point detector, and a clustering procedure. The clustering procedure identifies jobs of similar characteristics; the change-point detector determines periods of stationarity for the jobs; and the percentile estimator calculates a quantile that serves as a bound on future wait time.

Kumar and Vadhiyar [22] developed a technique that defines which jobs can be classified as *quick starters*. These are jobs with short waiting times compared to the other jobs waiting for resources. Their technique considers both job characteristics such as request size and estimated runtime, and the state of the system, including queue and processor occupancy states.

C. Runtime predictions

Smith [12] developed a method/system for estimating both queue wait time and job runtime predictions. The method is based on instance learning techniques and leverages genetic algorithm to refine input parameters for the method to obtain more accurate predictions. This system is used by XSEDE[2] to predict queue wait time.

Yang et al. [23] proposed a technique to predict the execution time of jobs in multiple platforms. Their method is based on data collected from short executions of a job and the relative performance of each platform.

Tsafrir et al. [24] developed a technique for scheduling jobs based on system-generated job runtime estimates, instead of using user provided estimates. For the runtime estimates, they analyzed several workloads from supercomputer centers and found out that users tend to submit similar jobs over a short period of time. Therefore, their estimations are based on the average time of the previous two actual job runtime values.

Compared to efforts in the literature, we use a learning algorithm to make job placement decisions for hybrid cloud environments. These environments are different from traditional grid or multi-clusters as the cloud component is elastic and typically contains slower inter-node network than the on-premise component. Furthermore, we considered and handle the uncertainties in the job wait time and runtime predictions to make job placement decisions by leveraging existing predicting tools. Next section contains details of opportunities and problems posed by HPC hybrid cloud environments.

III. Problem Description

Due to the heterogeneity of jobs in several supercomputing settings, mixing on-premise and cloud resources is a natural way to get the best of the two environments. In hybrid HPC clouds, users can experience fast interconnections in on-premise clusters[3] and quick access to resources in the cloud. Hybrid clouds are also cost-effective since it is possible to keep a certain amount of on-premise resources and rent cloud resources to meet minimal and peak demands, respectively.

With this hybrid environment one major challenge to users is to know where they should run their jobs. The decision involves several factors such as costs, raw performance on both environments, resource provisioning time in the cloud, and queue waiting time in the on-premise clusters. In this paper we envision a scenario where a company or supercomputing center has a contract with a cloud provider, which makes the cost to access cloud resources transparent to end-users, whose main concern is to optimize their jobs' turnaround time. However, the findings of this paper can be incorporated into other scenarios where users are totally exposed to cloud costs, or users receive quotas to use cloud resources; therefore they want also to minimize monetary costs.

[2]XSEDE - https://www.xsede.org/

[3]Fast network interconnects, e.g. InfiniBand, are still not popular in public clouds.

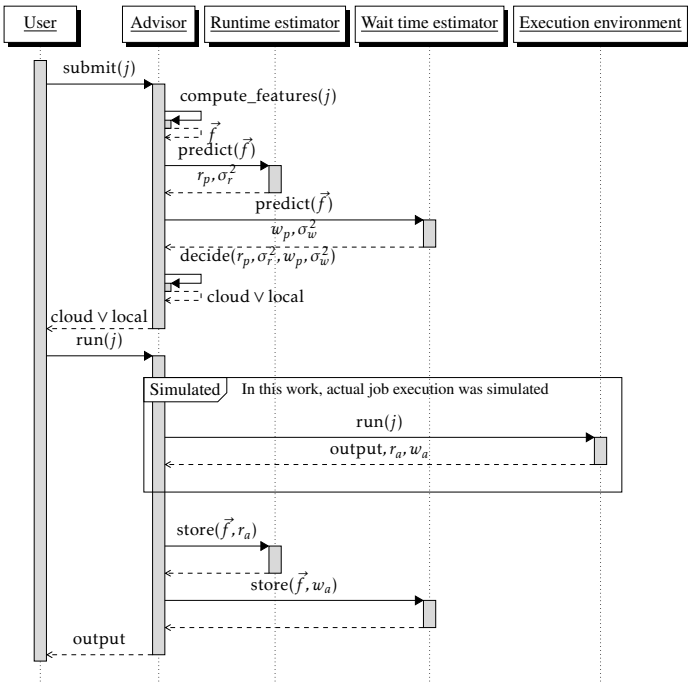

Figure 1. Sequence diagram of operations performed by the advisor when a job is submitted by a user. After predicting the run and wait times for a new job, the advisor presents its decision to the user for validation. The user, then selects the execution environment, and the advisor submits the job for execution, storing actual wait and runtimes in the estimators as soon as the job is finished.

The decision where to run a particular job would be simplified if execution times and cluster waiting times were known in advance and were accurate. In practice this is not possible since (i) job execution times depend on application input and underlying computing infrastructure, and (ii) waiting times depend on unpredictable scheduler queue dynamics created by other job arrivals and completions.

The problem we tackle in this paper is therefore: *"how to predict job execution times and job waiting times, and more importantly, how to make job placement decisions based on the fact that these predictions can be inaccurate"*. We rely on state-of-the-art statistical models to make predictions, which are based on knowledge of past jobs and their characteristics.

Our scenario therefore consists of a company/institution with a set of resources controlled by a cluster management system (such as LSF, SLURM, TORQUE, and PBS) and a cloud provider. We explore scenarios with different performance ratios between both environments. The cluster management system contains records of already executed jobs. In our case we use traces from supercomputers available in the Parallel Workloads Archive [25]. We assume data-intensive jobs already have their data in cloud to be processed. For cases that require time transfer data to the cloud, the techniques proposed here can be further enhanced to include such a time.

IV. Advisor tool: overview and method

The focus of this work is on developing and evaluating a decision-support tool, the *advisor*, built using prediction techniques from the literature [12]. These techniques are based on IBL services that provide two types of predictions: wait time estimates (how long a job is expected to wait in a resource manager queue before execution) and runtime estimates (for how long a job is going to run, once it starts execution).

The advisor acts as a traditional resource management system: the user submits a job to the advisor, which calculates the best environment for job execution, and then forwards jobs to the actual resource manager. If the user disagrees with the advisor's suggestion, she can still force her job to execute in her chosen environment [26].

Figure 1 shows a sequence diagram of the major steps involved in job submission. Once the advisor receives a new job submission request j, it extracts features $\vec{f}$ from it, and predicts the time j would be expected to wait (w_p) should it be scheduled to run in the on premise cluster, and the expected time it would take for j to run (r_p) in the same environment[4]. The output of each prediction is a pair *expected time, associated uncertainty*. The uncertainty is represented with a variance measure, and is denoted as σ_w^2 for the wait time predictor, and as σ_r^2 for the runtime predictor. As soon as the user makes her decision (either by accepting the advisor's suggestion or by forcing her preference), the job is submitted to the execution environment.

The decide function shown in Figure 1 takes the outputs of the runtime (r_p and σ_r^2) and wait time estimators (w_p and σ_w^2) and decides whether a job should be run locally or on the cloud. It decides to send a job to the cloud whenever

$$\left(w_p + r_p < (oj_c + i)\, r_p\right) \wedge \sigma_w^2 < \mathrm{C}\left(o, \vec{f}\right)$$

is true, where o is the penalty for each additional processor used on the cloud (representing communication *overhead*) and i is the baseline, j_c is the number of processors requested in job j, C defines a variance *cutoff* value based on job characteristics and the communication penalty. If the wait time variance is too high, this function returns false and the job is scheduled to run locally. The cutoff is dynamic and depends on the job features and on o. The rationale behind this decision is to guide the advisor to make more conservative decisions, favoring the local environment when estimates are too uncertain. In this work, we decided to only consider the wait time variance for the cutoff. As will be discussed later, estimating the wait times of jobs is harder than estimating their runtimes and, thus, this variance accounts for most of the uncertainty of predictions.

A. Estimator design

This section describes the implementation of the estimators used in this work. For a discussion of the rationale behind the choice of methods and functions, we refer the reader to the work we based this on [12].

As mentioned previously, we use an IBL technique for implementing the estimators. IBL techniques work by constructing hypotheses based on previous examples stored in a knowledge

[4]The ordering of these calls is irrelevant, and these calls can be made in parallel, as processing only continues *after* both predictions are made.

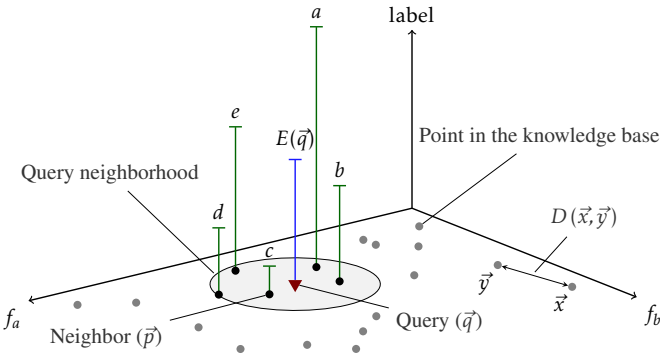

Figure 2. Overview of the k-NN method used in this work. Axes f_a and f_b represent the features used, and the label axis represents the *values* recorded for those features (one notable exception is $E(\vec{q})$, which is the *predicted* value for the query, computed from the neighbors a, b, c, d, and e). The points in black constitute the query point's neighborhood, while the points outside the neighborhood are represented in gray and without associated labels, to avoid polluting the image. Note that the radius of the neighborhood is not fixed, and will always be equal to the distance of the farthest neighbor of $\vec{q}$.

base, typically created in a training phase of the algorithm. The data stored in the knowledge base has two parts: a set of *input features* and a set of *output features*. A feature is a variable that has a name and a type, and holds a value. Types usually are strings (nominal features), integers or floating-point numbers, and a feature's value belongs to the feature's type. Input features describe the conditions under which an event happened, and the output features describe what happened, in a dimension of interest, under those conditions.

In this work we used an algorithm known as k-Nearest Neighbors (k-NN). The principle behind this algorithm is, given a query, to find a predefined number of training samples closest in distance to the query and, then, to make a prediction using the k nearest points found, as shown in Figure 2. To compute distances we used the Heterogeneous Euclidean-Overlap Metric (HEOM) [27]. This function defines the distance between two values x and y of a given feature f as

$$d_f(x,y) = \begin{cases} 1 & \text{if } x \text{ or } y \text{ is unknown,} \\ \text{overlap}_f(x,y) & \text{if } f \text{ is a nominal feature,} \\ \text{diff}_f(x,y) & \text{otherwise} \end{cases}$$

with the overlap_f and the diff_f functions defined as

$$\text{overlap}_f(x,y) = \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{otherwise} \end{cases}$$

$$\text{diff}_f(x,y) = \frac{|x-y|}{\max_f - \min_f}$$

where $\max_f$ and $\min_f$ are, respectively, the maximum and minimum values observed in the knowledge base[5] for feature f.

The above definition returns a value that is typically in the range $[0,1]$, whether the feature is nominal (in our case, a category number), or a number. With these definitions, the overall distance between two vectors of features $\vec{x}$ and $\vec{y}$ is defined as

$$D(\vec{x},\vec{y}) = \text{HEOM}(\vec{x},\vec{y}) = \sqrt{\sum_{f=1}^{N} w_f d_f(x_f, y_f)^2} \qquad (1)$$

where x_f is the scalar value of the f-th component of vector $\vec{x}$, N is the total number of features, and w_f is the feature's weight. Adding weights to features allows us to tune the algorithm to make more important features influence more the prediction.

The distance between two points, obtained with the D function, is passed into the kernel function, which weights the values of the distances. Such weighting happens in a way that, as distances approach zero, their weights approach a maximum constant value, and as distances approach infinity, their weights approach zero. In this work we are using a Gaussian function as a kernel. This function also includes a parameter, denoted by ω, that defines the stretch of the Gaussian, increasing or lowering weights of points that are farther away. Therefore, the kernel function is defined as

$$K(d) = e^{-\left(\frac{d}{\omega}\right)^2}. \qquad (2)$$

At this point, we have all the necessary building blocks for defining the estimation function, which computes a distance-weighted average of the output values of the k nearest neighbors of a query point $\vec{q}$. The intuition behind the distance-weighted average is that of spacial locality: examples more similar to a query should yield more similar results. Hence, the estimation function is defined as

$$E(\vec{q}) = \frac{\sum_{\vec{p}} K\Big(D(\vec{q},\vec{p})\Big)\, l(\vec{p})}{\sum_{\vec{p}} K\Big(D(\vec{q},\vec{p})\Big)} \qquad (3)$$

where $\vec{p}$ is a neighbor of $\vec{q}$, $l(\vec{p})$ is the output value of $\vec{p}$, and $\sum_{\vec{p}}$ is a sum over all neighbors.

To measure the uncertainty of a prediction, we compute the unbiased estimate of its variance, given as

$$s^2(\vec{q}) = \frac{\sum_{\vec{p}} K\Big(D(\vec{q},\vec{p})\Big)\big(E(\vec{q}) - l(\vec{p})\big)^2}{-1 + \sum_{\vec{p}} K\Big(D(\vec{q},\vec{p})\Big)} \qquad (4)$$

and output it together with the estimate $E(\vec{q})$.

The above discussion defines how predictions can be made assuming a knowledge-base already exists. The initial version of the knowledge base is constructed in the training phase of the algorithm, but we need a way to *update* the model. Otherwise, as time passed, the data in the model would lose temporal locality, and would probably generate noisier predictions. The update step should work by removing older points from the knowledge base, which would become obsolete from a temporal locality standpoint, and by adding newer ones, as new queries would be closer in time to them. Due to that, and for simplicity reasons, we use a simple FIFO replacement policy for updating the model: every time a new experience is added to the knowledge base, the oldest one is removed from it.

[5] This means it is possible to obtain a value greater than one when computing the distance for a given feature.

Table I
INPUT FEATURES USED IN THE ESTIMATORS.

Feature	Type	Description
User ID	Category	The user that submitted the job
Group ID	Category	The group of the user that submitted the job
Queue ID	Category	The number of the queue the job has been submitted to
Submission Time	Number	The time at which the job was submitted.
Requested Time	Number	The amount of time requested to execute this job
Queue Size	Number	The number of queued jobs at the moment of submission of a single job
Queued Work	Number	The amount of work that was in the queue at the moment of submission of a single job
Remaining Work	Number	The amount of work remaining to be executed at the moment of submission of a single job
Weekday	Number	The day of the week in which a job was submitted
Time since midnight	Number	The time of the day at which a job was submitted
Free processors	Number	The amount of free processors in the cluster when this job was submitted

Table II
OUTPUT FEATURES USED IN THE ESTIMATORS.

Feature	Description
Estimated time	The estimated time as computed by (3).
Estimation variance	The associated variance computed by (4).

The estimator used in this work relies on parameters obtained by a genetic algorithm (GA) stochastic search. Different workloads may require different parameters, as the performance of the estimator varies substantially depending on the selected parameters. For optimization, we defined a GA chromosome as $\left[k, \omega, s, w_{f_1}, \ldots, w_{f_n}\right]$, where k is the number of neighbors, ω is the weight parameter of the kernel function, s is the size of the knowledge base, and w_{f_1} to w_{f_n}, used in (1), are the weights of features f_1 to f_n. The input features we used are described in Table I. These features are computed at the moment of submission of a single job and do not change once they have been computed. This same set is used both when predicting wait times and runtimes.

The scoring function used in the GA is the Root Mean Squared Error (RMSE) function, defined as

$$\text{RMSE}(\hat{y}, y) = \sqrt{\frac{\sum_{t=1}^{N} (\hat{y}_t - y_t)^2}{N}} \tag{5}$$

where N is the number of samples (predictions made), y_t is the real value of a sample (in the runtime predictor, the real execution time, and in the wait time predictor, the real time a job waited in the queue), $\hat{y}_t$ is the value predicted. The objective of the GA is to find the best individuals that minimize the RMSE. The input to the RMSE function are $(\hat{y})$ an array of estimated times from the output features, described in Table II[6], and (y) an array of the actual execution times of the same jobs for which the predictions were made. When used in the runtime estimator, the output features are runtime estimates and, similarly, when used in the wait time estimator, they are estimated wait queue times.

[6]With the "type" column ommited, since output features have the same floating-point numeric type.

B. Correctness issues when using floating-point arithmetic

The estimator from (3) has a major correctness issue: it has a non-zero probability of making divisions by zero. This derives from the fact that any number smaller than $f_{\min}$, the smallest representable number in a floating-point system, is represented as zero [28]. Hence, when

$$e^{-\left(\frac{d}{\omega}\right)^2} < f_{\min},$$

(2) will be equal to zero. From this, it follows that when

$$\frac{d}{\omega} > \sqrt{-\ln(f_{\min})},$$

$K(d) = 0$. In an IEEE-754 environment with double precision (normalized and without gradual underflow), $f_{\min} \approx 2.23 \times 10^{-308}$ and the kernel will be zero whenever $d/\omega \gtrapprox 26.62$.

A direct consequence from the result above is that whenever a query has a feature value greater than the maximum value stored in the knowledge base, d will be greater than or equal to one and that feature will be zeroed whenever $\omega < 1/\sqrt{-\ln(f_{\min})} \approx 0.038$. If, for some reason, a new query is sufficiently far from all its neighbors (i.e. $d \geq 1$), then $\sum_{\vec{p}} K\left(D(\vec{q}, \vec{p})\right) = 0$, and the estimator from (3) will generate $0/0$, a case that must be handled. This case can happen easily in the GA optimization step, which tries random values for members of the chromosome array, but we have also seen this happen in the prediction step with already optimized estimators.

In cases where NANs happened during the GA optimization, we defined the RMSE of individuals that generated NANs to be a large value, to penalize similar solutions. A potential drawback of this approach is that in cases where almost all predictions had low errors, but a single prediction yielded a NAN, the whole solution would be discarded. The dual of this approach, i.e. counting NANs as zero error, would favor solutions with NANs, undermining the optimization procedure.

When NANs are generated by the estimators at the prediction phase, the advisor's decide function makes a conservative decision and returns a suggestion to run the job locally, as no certain statements can be made about jobs that make NANs happen.

V. Evaluation

The evaluation of the advisor is presented in this section. The estimator implementation is validated and metrics and workloads are described. The evaluation considers a theoretical model of speedup ratios and real speed up ratios from various supercomputers and cloud providers available in the literature of HPC cloud.

A. Estimator validation

The estimators play a key role in the advisor and it is essential to validate our implementation compared to estimators found in the literature. Therefore, in order to validate our implementation, we evaluated it using the original data from Smith [12]: anonymized queue logs from the TACC Lonestar system.

For training, we used 500 log entries starting from February 2006. The knowledge base started from the beginning of the log until the last entry that had started before the submission time of the first entry in the testing set. The GA was run with a population of size 50 for 100 generations. The probability of crossover was 80% and the probability mutation 10%. Elitism was set to 5%.

Once the weights were obtained, we ran a prediction phase with unseen data from a posterior section of the data log. The validation phase had $N = 10000$ entries. In order to evaluate the prediction, we used the mean error, that is defined as follows:

$$\frac{\sum_{i=0}^{N-1} |\,\text{actual}_i - \text{predicted}_i\,|}{N}$$

For execution time, we obtained a mean error of 125.99 minutes. This value is similar to the result obtained by Smith [12], i.e. 122.44 minutes.

B. Evaluation metrics

Once validated, the next step in the evaluation process was to test the performance of the advisor. We used logs from the Parallel Workload Archive [25], since they are open, freely available, and can be used by anyone willing to validate our results. Specifically, we used the logs of San Diego Supercomputer Center's (SDSC) Blue Horizon (BLUE) and DataStar (DS) clusters, and from the High-Performance Computing Center North's Seth cluster (HPC2N). As in the estimator's validation, for all these workloads we used 500 log entries for GA optimization. We also discarded the first 10000 entries of the logs, to prevent any cold-start effects, and once the estimators were trained, we evaluated them using 10000 more log entries.

A possible metric for evaluation could be to still use RMSE and evaluate its value as we applied the cutoff function to the uncertainty. Intuitively, one could argue that as we remove less certain predictions, the RMSE drops. However, this metric does not represent the real objective for the advisor, as in this case there is a trivial solution: make the cutoff so low that no job goes to the cloud. Consequently the RMSE would be made equal to zero, and an "optimal" solution would be achieved.

Another possible metric to use could be accounting for hits and misses, which correspond to predictions that are confirmed or refuted, respectively, by the real waiting time and execution time. Whenever the advisor tells the user to send a job to the cloud based on the waiting time and execution time predictions and the actual execution time in the cloud is smaller than the local turnaround time, this counts as a hit, otherwise it is a miss. The same procedure would be applied when the advisor tells the user to go to the on-premises resources. Such a metric may be misleading though, since the advisor may perform well with small gain jobs, i.e. jobs whose saved time is small, and perform badly with jobs that cause large losses, giving the illusion of good performance without that being the case.

A better metric, and the one we settled on, is to evaluate the time saved when submitting jobs to the cloud. If the advisor tells the user to send a job to the cloud, we evaluate the time it takes for the job to execute in the cloud and subtract this value from the actual waiting time plus the actual execution time. Our base case for comparison is the *always-local* strategy, which represents a conservative user that runs jobs only locally. Therefore, if the advisor decides to run a job locally, the time it saved is equal to zero:

$$T = \begin{cases} r^{\text{cloud}} - \left(r^{\text{local}} + w_a\right) & \text{if job runs in the cloud,} \\ 0 & \text{otherwise} \end{cases}$$

where r^{cloud} is the runtime in the cloud, r^{local} is the runtime in the local environment, and w_a is the wait time in the local environment. T can be negative and that represents losses compared to the conservative strategy.

C. Analysis using theoretical speedup ratios

The prediction of on-premise turnaround time depends on the prediction of both waiting time and execution time. These predictions are not perfect, though, as shown in Figure 3, which displays a scatter plot of actual and predicted runtime and waiting time. Particularly, waiting time has bigger errors than runtime due to highly-varied queue dynamics, which directly influence queue wait times. Consequently, waiting time plays a more significant role in the uncertainty of the advisor's decision, and this is the reason for using a cutoff function that only takes the waiting time variance into account.

Aside from the predictions, the advisor needs to evaluate the relative performance speeds of on-premise and cloud resources. The execution time in the on-premise resources can readily be obtained from the logs and corresponds to the *Run Time* field in the Standard Workload Format [25]. As for the cloud, the execution time depends on many factors. Differences in network interconnects play a dominant role in performance differences, since clouds usually do not employ HPC-optimized networks. From this, it follows that as jobs with larger numbers of processors are submitted to the cloud, their performance deviate more from that seen in on premises clusters. In our analysis, we model the performance of the cloud as an affine function of the number of requested processors:

$$r^{\text{cloud}}(j_c) = (o\,j_c + i) \cdot r^{\text{local}}$$

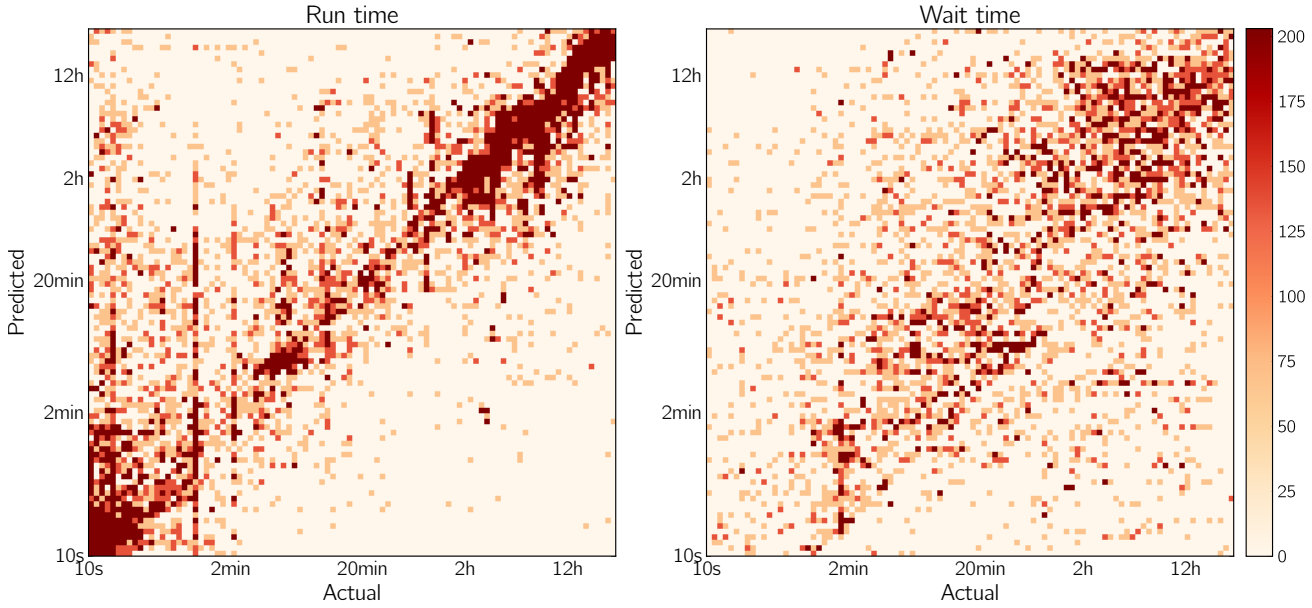

Figure 3. Log log scatter plots of runtime predictions and wait time predictions for the HPC2N workload. Darker areas are more densely populated. Lighter areas are less densely populated. As can be inferred, runtime estimations are more accurate than wait time estimations.

where j_c is the number of used processors, o is the overhead factor for each additional processor, and i is the runtime component that is independent of the number of processors. Even though this is a simple model, it can be adjusted as the user submits jobs to the cloud. For our evaluations, this model corresponds also to the model of the reality, i.e. we do not adjust this model as applications are submitted to the cloud. The reason for this simplification is that we can isolate the evaluation of the advisor from the cloud model, which is a learning procedure in itself.

The advisor's decisions for different overhead factors are presented in Tables III, IV and V. For each log, we analysed the time-saved without the cutoff in the variance and with an exponential cutoff as a function of the overhead factor. The reason for doing so is that as the cloud becomes slower compared to the local environment, the user tolerates less uncertainty, because decision errors would waste more time. However, for more certain predictions the user might still want to submit jobs to the cloud. We compare the results with an optimal advisor (shown in the last three columns), which corresponds to the behavior of the advisor if it knew in advance the actual waiting and execution times. Notice that even with very high factors, some jobs can still be sent to the cloud, as shown in the "# of jobs on cloud" column of the optimal allocation.

There are some key differences among the analyzed systems. The jobs in the DS cluster are heavily penalized by slower clouds, while jobs from the HPC2N are not. In addition, when using the BLUE workload, the advisor presents its worst performance in the middle range of factors. These facts reflect the different characteristics of the jobs and how these differences influence the advisor. For example, more than 80% of jobs of DS are short jobs (less than 1 hour), even though only 30% have requested times that are less than 1 hour. While in HPC2N there are longer jobs and requested times are more similar to actual runtimes. These facts help the predictor and consequently the results are better for HPC2N. As for the BLUE workload, decisions are only affected when the overhead factor is sufficiently high, influencing the advisor to reduce the number of jobs sent to the cloud. In the middle range of factors, a sufficiently large number of jobs is still sent to the cloud, which accounts for the losses seen in the table.

D. Analysis using real speedup ratios

In this section, we evaluate the advisor using real factors extracted from the work of Gupta et al. [29]. In that work, they compared the runtime of 8 scientific applications on 3 cloud and 3 local environments. Our objective here is to have a better approximation of the performance of the advisor with more realistic speed ratios between cloud and local.

Table III
JOB ALLOCATION DECISIONS AND SAVED TIME FOR THE SDSC BLUE CLUSTER.

Factor	Without cutoff function			With cutoff function			Optimal allocation		
	# of jobs on-premises	# of jobs on cloud	Saved-time (millions of sec.)	# of jobs on-premises	# of jobs on cloud	Saved-time (millions of sec.)	# of jobs on-premises	# of jobs on cloud	Saved-time (millions of sec.)
0.01	4227	4318	130.05	4250	4295	123.9639	4758	3787	155.98
0.05	6675	1870	12.54	7049	1496	2.2067	6437	2108	80.34
0.10	7383	1162	-1.86	7776	769	0.3998	7118	1427	55.21
0.20	7904	641	-10.71	8421	124	0.0124	7607	938	40.86
0.30	8083	462	-12.02	8534	11	0.0005	7791	754	35.57
0.40	8219	326	-4.02	8545	0	0.0000	7929	616	32.49
0.50	8272	273	-2.18	8545	0	0.0000	8002	543	30.46
0.75	8362	183	-3.04	8545	0	0.0000	8100	445	26.99
1.00	8407	138	-3.97	8545	0	0.0000	8173	372	24.72
10.00	8531	14	-0.40	8545	0	0.0000	8446	99	10.21

Table IV
JOB ALLOCATION DECISIONS AND SAVED TIME FOR THE SDSC DS CLUSTER.

Factor	Without cutoff function			With cutoff function			Optimal allocation		
	# of jobs on-premises	# of jobs on cloud	Saved-time (millions of sec.)	# of jobs on-premises	# of jobs on cloud	Saved-time (millions of sec.)	# of jobs on-premises	# of jobs on cloud	Saved-time (millions of sec.)
0.01	1859	4692	131.95	1868	4683	130.9516	2162	4389	177.75
0.05	3746	2805	27.44	4182	2369	11.6993	3809	2742	134.95
0.10	4316	2235	-37.27	5282	1269	0.8732	4393	2158	110.72
0.20	4773	1778	-135.92	5910	641	-4.5077	4921	1630	86.88
0.30	5092	1459	-223.93	6307	244	-3.5278	5211	1340	74.38
0.40	5314	1237	-266.48	6515	36	-4.7084	5442	1109	66.42
0.50	5444	1107	-339.40	6549	2	-0.0060	5573	978	61.11
0.75	5714	837	-435.65	6551	0	0.0000	5804	747	54.14
1.00	5847	704	-551.68	6551	0	0.0000	5902	649	50.26
10.00	6341	210	-1394.15	6551	0	0.0000	6289	262	24.97

Table V
JOB ALLOCATION DECISIONS AND SAVED TIME FOR THE HPC2N CLUSTER.

Factor	Without cutoff function			With cutoff function			Optimal allocation		
	# of jobs on-premises	# of jobs on cloud	Saved-time (millions of sec.)	# of jobs on-premises	# of jobs on cloud	Saved-time (millions of sec.)	# of jobs on-premises	# of jobs on cloud	Saved-time (millions of sec.)
0.01	2659	7319	360.70	2659	7319	360.7023	3497	6481	402.86
0.05	4612	5366	276.48	4634	5344	270.8140	4928	5050	362.47
0.10	5669	4309	200.13	6313	3665	107.6861	5649	4329	328.30
0.20	6617	3361	134.86	8406	1572	28.4573	6391	3587	287.36
0.30	7170	2808	87.45	9210	768	11.3742	6841	3137	262.54
0.40	7547	2431	61.50	9737	241	3.7432	7119	2859	244.13
0.50	7810	2168	42.07	9797	181	5.7405	7318	2660	229.89
0.75	8231	1747	12.23	9837	141	3.3188	7692	2286	205.51
1.00	8459	1519	3.88	9851	127	4.9757	7941	2037	189.75
10.00	9579	399	-49.85	9933	45	0.1299	9105	873	83.65

From that data [29] we extracted performance ratios in the form

$$s(j_c) = \frac{r_a^{\text{cloud}}(j_c)}{r_a^{\text{local}}(j_c)}$$

where r_a^{cloud} and r_a^{local} are the runtimes of a given application on cloud and local environments, respectively. For each triple (cloud, local, application), we have a different $s(j_c)$, totalling $8 \cdot 3 \cdot 3 = 72$ different performance ratios. Since that work only measured runtimes using number of processors (j_c) in the range $[1, 256]$, we limited our analysis to jobs in that range.

Because we are working with *ratios*, the new model for predicting turnaround time is defined as

$$r_p^{\text{cloud}} = s(j_c) \cdot r_p^{\text{local}}$$

where r_p^{cloud} and r_p^{local} are the predicted runtimes of cloud and local environments respectively.

Next, we used the advisor to select the environment where to run jobs and, as in the previous section, we computed the time saved. Figure 4 shows the results for the three workloads used. The graphs show time saved as a function of the scenario, where each scenario correspond to a triple (cloud, local, application)—

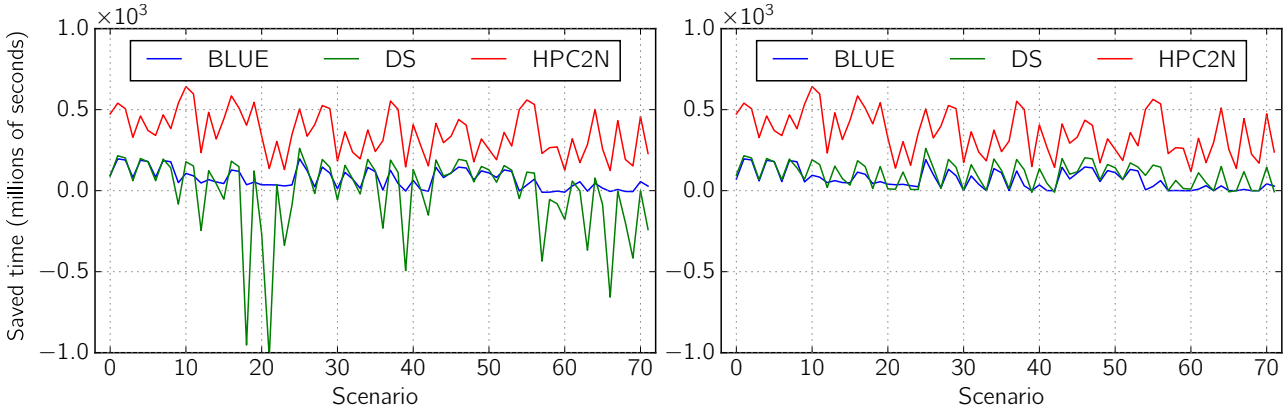

Figure 4. Saved time for all 72 scenarios with and without the cutoff function.

notice, however, that there is no particular ordering between scenarios. The graph in the left side displays the results without the cutoff function, while the graph on the right shows time saved with an exponential cutoff function of the number of processors used.

Without the cutoff function, decisions for the DS workload wasted time for many scenarios, while in the BLUE and HPC2N workloads, time was saved for almost all scenarios. In order to clarify what caused those results we plotted the average ratio between cloud and local for each one of the scenarios (Figure 5). The average ratio used in HPC2N is much smaller than in the others, since most of the jobs in HPC2N used few processors. Consequently, the ratios are small and the penalty for eventual allocation errors does not affect much the time saved metric. This, however, does not explain the difference between DS and BLUE. For these two workloads, the reason for the difference lies on the fact that the predictions for BLUE overestimate the wait time of the jobs. Consequently, more jobs are sent to the local environment. As a result, the advisor becomes more conservative and its behavior is closer to the always-local strategy. In the DS workload, predictions do not have a clear pattern, and the advisor's mistakes outweighs the times in which it makes correct decisions, resulting in unsatisfactory performance. Nonetheless, with the addition of the cutoff function, the advisor is able to decide with greater certainty when to run jobs in the cloud, as one can see in the right side of Figure 4. This reflects the fact that, for high ratio values, the advisor takes a conservative approach and becomes similar to the always-local strategy.

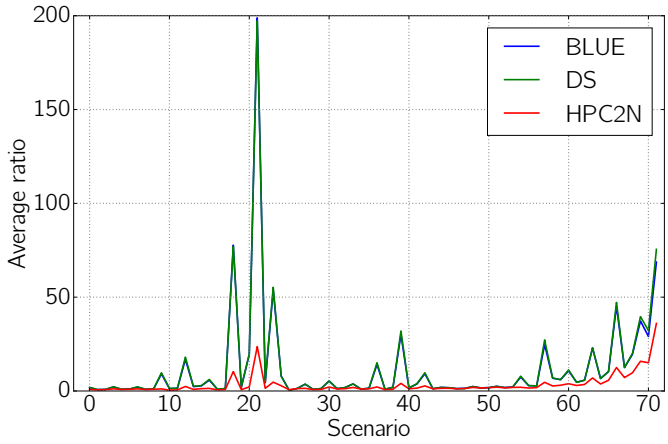

Figure 5. Average ratio as function of scenario

VI. Conclusion

In this paper, we proposed an advisory tool to help users decide where to run jobs in a hybrid HPC cloud environment. The decision depends on queue waiting time and execution time of the jobs, which are predicted using traces from past job scheduling data. Even though we have used state-of-art predictors, the estimations of wait time and runtime can be inaccurate. Therefore, some strategy is required to handle poor estimations and avoid delaying job turnaround times.

One of our contributions is to define a cutoff function of the uncertainty, that limits the impact of prediction errors in the time saved when using cloud resources. This cutoff establishes the maximum uncertainty tolerated by the user to accept predicted results. Beyond this value the advisor takes a conservative approach of submitting a job to on-premise resources. The function is parametrized by a speed factor. The logic behind it is that the user tolerates less uncertainties when the cloud is slower and, consequently, the penalty for errors are larger.

Some lessons learned in this work are: (1) simply increasing the knowledge base's size with more past data may decrease the performance of the predictor; (2) using jobs identified as quick starters can actually make predictions worse; (3) even though cleaned, logs can still have many inconsistencies that need to be fixed, for example, incomplete features; and (4) numerics, such as the precision of floating-point arithmetic may play a role in the results. Some of these lessons were discussed in the text, but others were not. However, we still enumerate them here as a warning to the reader.

Finally, we observed that the demand for hybrid clouds has been growing at a fast rate. The work presented in this paper

can be effectively used by the HPC community, even if cloud resources are not as powerful as local ones. The work presented here aims to help users to make better resource allocation decisions, but we believe there is still room for improvements. In particular, better predictors can improve the confidence of users on where to place their jobs and consequently increase the demand for this type of HPC hybrid cloud services.

ACKNOWLEDGEMENTS

We thank Warren Smith for providing anonymized data used to validate our estimator implementation.

REFERENCES

[1] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Cloudcom*, 2010.

[2] J. Napper and P. Bientinesi, "Can cloud computing reach the top500?" in *UCHPC-MAW*, 2009.

[3] C. Vecchiola, S. Pandey, and R. Buyya, "High-Performance Cloud Computing: A View of Scientific Applications," in *ISPAN*, 2009.

[4] A. Gupta, L. V. Kale, F. Gioachin, V. March, C. H. Suen, B.-S. Lee, P. Faraboschi, R. Kaufmann, and D. Milojicic, "The who, what, why and how of high performance computing applications in the cloud," in *CloudCom*, 2013.

[5] E. Roloff, M. Diener, A. Carissimi, and P. O. A. Navaux, "High performance computing in the cloud: Deployment, performance and cost efficiency," in *CloudCom*, 2012.

[6] K. Keahey and M. Parashar, "Enabling on-demand science via cloud computing," *IEEE Cloud Computing*, 2014.

[7] K. Mantripragada, L. P. Tizzei, A. P. D. Binotto, and M. A. S. Netto, "An SLA-based advisor for placement of HPC jobs on hybrid clouds," in *ICSOC*, 2015.

[8] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in *CloudCom*, 2010.

[9] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *PDP*, 2013.

[10] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, and X. Yuan, "A comparative study of high-performance computing on the cloud," in *HPDC*, 2013.

[11] I. Sadooghi, J. Hernandez Martin, T. Li, K. Brandstatter, Y. Zhao, K. Maheshwari, T. Pais Pitta de Lacerda Ruivo, S. Timm, G. Garzoglio, and I. Raicu, "Understanding the performance and potential of cloud computing for scientific applications," *IEEE Transaction on Cloud Computing*.

[12] W. Smith, "Prediction services for distributed computing," in *IPDPS*, 2007.

[13] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, 1991.

[14] M. D. De Assunção, A. Di Costanzo, and R. Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *HPDC*, 2009.

[15] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan, "Scheduling of parallel jobs in a heterogeneous multi-site environment," in *JSSPP*, 2003.

[16] S. Sotiriadis, N. Bessis, and N. Antonopoulos, "Towards inter-cloud schedulers: A survey of meta-scheduling approaches," in *3PGCIC*, 2011.

[17] N. Bessis, S. Sotiriadis, F. Xhafa, F. Pop, and V. Cristea, "Meta-scheduling issues in interoperable HPCs, grids and clouds," *International Journal of Web and Grid Services*, 2012.

[18] S. K. Garg, R. Buyya, and H. J. Siegel, "Scheduling parallel applications on utility grids: time and cost trade-off management," in *ACSE*, 2009.

[19] M. Malawski, K. Figiela, and J. Nabrzyski, "Cost minimization for computational applications on hybrid cloud infrastructures," *Future Generation Computer Systems*, 2013.

[20] H. Li, J. Chen, Y. Tao, D. Gro, and L. Wolters, "Improving a local learning technique for queuewait time predictions," in *CCGrid*, 2006.

[21] D. Nurmi, J. Brevik, and R. Wolski, "QBETS: queue bounds estimation from time series," in *JSSPP*, 2008.

[22] R. Kumar and S. Vadhiyar, "Identifying quick starters: towards an integrated framework for efficient predictions of queue waiting times of batch parallel jobs," in *JSSPP*, 2013.

[23] L. T. Yang, X. Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution," in *SC*, 2005.

[24] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems*, 2007.

[25] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *Journal of Parallel and Distributed Computing*, 2014.

[26] M. A. S. Netto, R. L. F. Cunha, and N. Sultanum, "Deciding When and How to Move HPC Jobs to the Cloud," *Computer*, 2015.

[27] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *Journal of Artificial Intelligence Research*, 1997.

[28] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, 1991.

[29] A. Gupta, P. Faraboschi, F. Gioachin, L. Kale, R. Kaufmann, B.-S. Lee, V. March, D. Milojicic, and C. Suen, "Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud," *IEEE Transactions on Cloud Computing*, 2014.