Characterizing a High Throughput Computing Workload: The Compact Muon Solenoid (CMS) Experiment at LHC

Rafael Ferreira da Silva[1], Mats Rynge[1], Gideon Juve[1], Igor Sfiligoi[2], Ewa Deelman[1], James Letts[2], Frank Würthwein[2], and Miron Livny[3]

[1] University of Southern California, Information Sciences Institute, Marina Del Rey, CA, USA
[2] University of California at San Diego, Department of Physics, La Jolla, CA, USA
[3] University of Wisconsin Madison, Madison, WI, USA
{rafsilva,rynge,juve,deelman}@isi.edu, {isfiligoi,jletts,fkw}@ucsd.edu, miron@cs.wisc.edu

Abstract
High throughput computing (HTC) has aided the scientific community in the analysis of vast amounts of data and computational jobs in distributed environments. To manage these large workloads, several systems have been developed to efficiently allocate and provide access to distributed resources. Many of these systems rely on job characteristics estimates (e.g., job runtime) to characterize the workload behavior, which in practice is hard to obtain. In this work, we perform an exploratory analysis of the CMS experiment workload using the statistical recursive partitioning method and conditional inference trees to identify patterns that characterize particular behaviors of the workload. We then propose an estimation process to predict job characteristics based on the collected data. Experimental results show that our process estimates job runtime with 75% of accuracy on average, and produces nearly optimal predictions for disk and memory consumption.

Keywords: High Throughput Computing, Workload Characterization, Job Characteristics Estimation

1 Introduction

Modern science often requires the processing and analysis of vast amounts of data and the validation of core principles through the simulation of complex system behaviors and interactions. As applications and infrastructure are growing in scale and complexity, understanding their behavior is a cornerstone for the development of efficient, reliable, and scalable systems.

The Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC) [4] enables physicists to conduct data analysis on the wide range of particles and phenomena produced in high-energy collisions in the LHC. Data analyses are performed on the CMS computing infrastructure, which uses HTCondor [19] as the underlying workload management system. It allocates and manages computing and storage resources for the execution of computational jobs. Currently, the infrastructure processes millions of jobs submitted by hundreds of users during month-long time frames. The efficiency of the workload execution and resource utilization strongly depends on how these jobs are scheduled and resources are provisioned. The scheduling problem is known to be an NP-complete problem [21], and thus many scheduling heuristics have been developed to address this issue [11,18,20]. However, most of these heuristics assume an accurate estimation of task characteristics, which in practice are hard to obtain. For instance, the scheduling algorithm used for the CMS experiment assumes a fixed runtime estimate for all of the users' jobs. Recently, Sfiligoi [16] analyzed CMS historical data to define the maximum and minimum runtime thresholds for predicting users' job runtimes. Although the analysis yields better estimates than user-provided estimates, job runtimes are not accurately estimated, and thereby resource utilization and workload efficiency are not optimal.

In this work, we first characterize and profile (§ 2) resource usage of jobs executed by the CMS experiment on a HTCondor pool. We apply statistical methods and machine learning techniques to discover patterns that describe specific behaviors of job executions (runtime, disk and memory usage). In particular, we extend the method proposed in our previous work [6]—where statistical correlations, clustering, and decision trees are used to estimate workflow task requirements—to identify statistical relationships between the workload properties, to build regression trees, and to identify probability distributions that would fit subsets of the workload (defined by regression tree leaves). Then, we propose a simple, yet practical, job estimation process (§ 3) based on the analysis of the regression trees and the probability distributions to estimate job requirements such as runtime, disk, and memory usage. Experimental results (§ 4) show that our process leads to nearly optimal estimates for disk and memory usage, and satisfactory accurate estimates for runtime when a small portion of the workload is used for training. High accuracy is also attained when estimating future workloads from historical data.

2 Characterization and Profiling of the CMS Data

Studies presented in the following sections are based on the workload of the HTCondor pool for the CMS experiment [2] deployed at the San Diego Supercomputing Center, for a period of a month (Aug 2014). Table 1 summarizes the main characteristics of the collected workload. The dataset consists of 392 users and 1,435,280 jobs, where 792,603 are completed jobs, 257,230 preempted jobs, and 385,447 have a non-zero exit code. Jobs ran on 15,484 execution nodes belonging to 75 different execution sites. For the sake of privacy, any user-specific data had been previously anonymized and not retained. Disk usage reports in HTCondor tend to be behind real time. Therefore, the disk usage recorded refers to snapshots of the disk usage at job runtime.

Characteristic		Data	
General workload	Total number of jobs		1,435,280
	Total number of users		392
	Total number of execution sites		75
	Total number of execution nodes		15,484
Jobs statistics	Completed jobs		792,603
	Preempted jobs		257,230
	Exit code (!= 0)		385,447
		Average	Std. Deviation
	Job runtime (in seconds)	9,444.6	14,988.8
	Disk usage (in MB)	55.3	219.1
	Memory usage (in MB)	217.1	659.6

Table 1: Characteristics of the CMS workload for a period of a month (Aug 2014).

2.1 Data Preparation

Real-world data may be incomplete, noisy, and inconsistent, which can obscure useful patterns [22]. For instance, profiling tools may not be able to collect and report the data once a job is preempted; or user defined scripts may alter application execution wrappers to attach an identification number for each execution. Data preparation is a fundamental stage of data analysis, which involves verifying the data accuracy; transforming the data; and developing and documenting a database structure that integrates the various measures. However, data preparation techniques cannot be fully automated, it is necessary to apply them with knowledge of their effect on the data being prepared [14]. In this work, we used our prior knowledge about the execution of scientific applications on HTC systems [7], and the knowledge of experts from the UCSD team to the data preparation process.

A typical CMS analysis consists of the execution of collision readout events, which are stored in files (approximately of the same size) logically grouped into datasets. In principle, all CMS analysis uses the same software base, CMSSW, but users may define their own code, analyses, etc. Nevertheless, this information cannot be gathered from the application execution program name or arguments in the logs, thereby the typical workload characterization by application name is not attainable. Each user submission is referred to as a *task*. The CMS Remote Analysis Builder (CRAB) [1] splits the task into a set of jobs (balanced by the number of events), and submits them to the HTCondor pool. Each job registers its task identification as the `blTaskID` property. This identification, however, has often a job identification number, appended to the task identification, for each job execution.

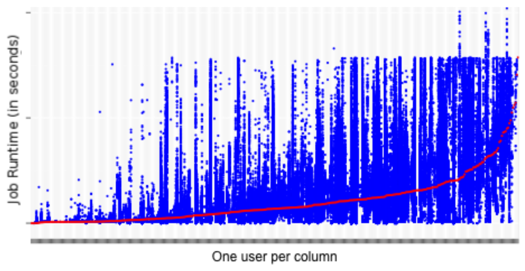
We assume that task identifications that follow a regular expression pattern are considered the same. Task identifications are considered similar if they match in at least 50%. This threshold is defined based on the knowledge about the data analyzed. For example, all task IDs in the raw data follow the `task_id_###` pattern, where `###` is often replaced by the job identification number or a random generated pattern. In this case, all matched patterns are replaced by the first matched pattern (e.g., `task_id_001`). Users are often logged as a path to the HTCondor log file on the submit host. This path contains the username and often a reference to the task ID. Thus, we also assume that usernames that follow a regular expression pattern are considered the same. The similarity threshold is also set to 50% based on the knowledge about the data analyzed.

2.2 Workload Execution Profiling

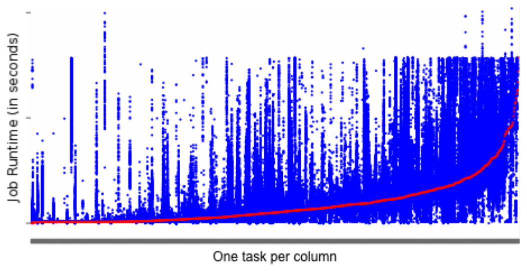
Although our workload consists of a 1 month period, it shows similar trends to the workload analysis conducted in [16], which was collected over a period of 5 months and limited to a few HTCondor queues. Correlations between users and job runtimes, as well as between a single task and job runtimes, are equivalent in both workloads. Figure 1 shows the distribution of job runtimes per user (Figure 1.a), and per task (Figure 1.b) for our workload. The mean value (red line in the graph) shows that the magnitude of the job runtimes varies among users and tasks. This result is thus used by the job estimation process (§ 3) to characterize the workload.

Figure 2 shows the distribution of job completion time rate. The workload has an average job arrival rate of 2,406 jobs per hour (standard deviation of 1,507), and an average job completion ratio of 2,438 jobs per hour (standard deviation of 1,457). The average job distribution per site $\bar{S}$ at an instant t is defined by the rate of the number of jobs allocated to a site and the number of execution nodes available in the site as follows:

$$\bar{S}(t) = \frac{1}{s} \times \sum_{i=1}^{s} \frac{j_i(t)}{h_i} \tag{1}$$



(a) Job runtimes by user, sorted by per-user mean job runtime in seconds.



(b) Job runtimes by task, sorted by per-task mean job runtime in seconds.

Figure 1: Job runtimes by user and task (red lines denote the mean value).

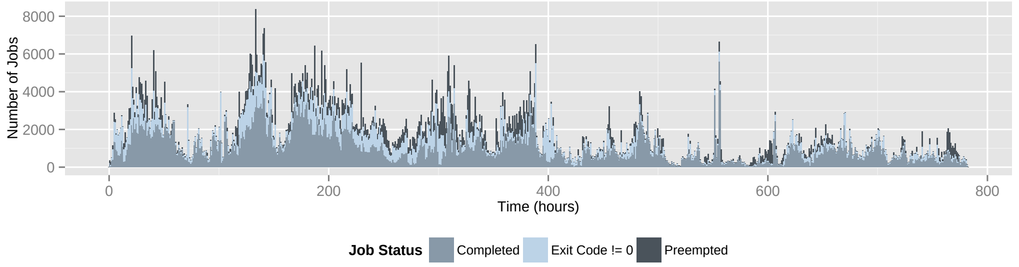


Figure 2: Job completion time rate of the CMS workload. The X-axis depicts the month long period covered by our dataset (Aug 2014). Colors represent different job status

where s is the number of sites, j_i the number of jobs allocated to a site i at instant t, and h_i the number of available execution nodes at site i. In the workload, the total average job distribution per site is $\bar{S} = 0.89$, which suggests that job distribution is relatively balanced among sites.

Although preempted jobs represent about 18% of the workload, they are not considered in this study, since they do not represent a complete execution and the data produced is insufficient. The analysis of the causes for preemption is an interesting research subject, but it is out of the scope of this work. Jobs with non-zero exit codes often represent error jobs, but some scientific applications may use non-zero exit codes to represent specific behaviors of successful executions. CMS job exit codes, however, represent execution errors [5], and thereby the analysis of these jobs is also out of the scope of this work.

2.3 Workload Characterization

In order to estimate job requirements, we first characterize the dataset to identify behavior patterns of the workload. Correlation statistics are enforced to the dataset to identify statistical relationships between variables. A variable is defined as one property of the dataset (e.g., `user`). Our dataset consists of 26 properties, thus the representation of each pair of correlations is not shown in this paper. Since the CMS experiment uses a single application for all users, no correlation can be determined between the property that defines the application (`command`) and any other variable. Weak correlations between the memory usage (`imageSize`), `diskUsage`, and job runtime (`duration`) with other variables suggest that none of the properties can be directly used to predict future workload behaviors. Trivial correlations as for example between `queueTime`

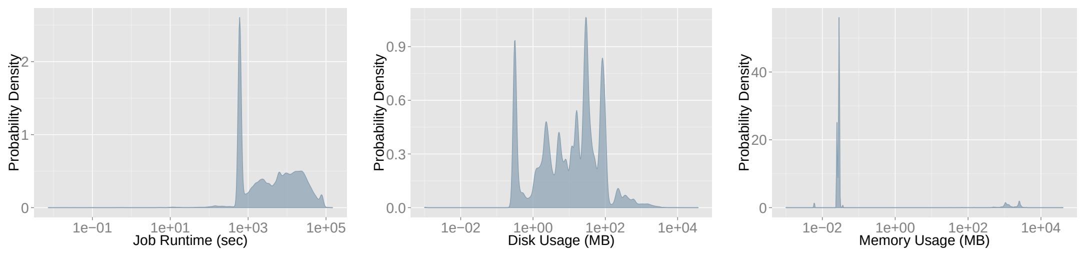


Figure 3: PDFs of job runtime (left), disk usage (middle), and memory usage (right).

and `startTime` or `completionTime`, or between `remoteUserCpu` and `duration`, for instance, are strong but do not provide any value for job characteristics prediction. However, most correlation measures are sensitive to the data distribution. Figure 3 shows the probability density function (PDF) of job runtime (left), disk usage (middle), and memory usage (right) for the workload. These probability functions do not fit any of the most common families of density such as the Normal or Gamma distribution, which can be characterized by unspecified parameters (e.g., mean and variance). Therefore, we explore the statistical recursive partitioning method to build decision trees, thereby dividing the dataset into smaller subsets.

The recursive partitioning method combines categorical (qualitative) and numeric (quantitative) properties from the workload to build regression trees where the variance of the predicted outcome (e.g., job runtime) is smaller than the original dataset. We use the results obtained in Figure 1 and in [16] to drive the partitioning method. Since the magnitude of job runtimes varies among users and tasks, we define the username as the root element of the regression tree. We then use the CTree (Conditional inference Trees) [9] algorithm to build the subsequent levels of the regression tree: the size of the application program (`executableSize`), the size of the input data (`inputsSize`), and the frequency the job restarted its execution due to (`numJobStarts`). Figure 4 shows an example of a regression tree for one user of the dataset. Terminal nodes show box plots of job runtimes. Although the standard deviation is smaller, the estimation of job runtimes, for instance, based on the average still leads to high estimation errors. The task identification (`blTaskID`) cannot be used here since each submitted task has a unique identification associated to it [16]. Note that a regression tree is built for each of the

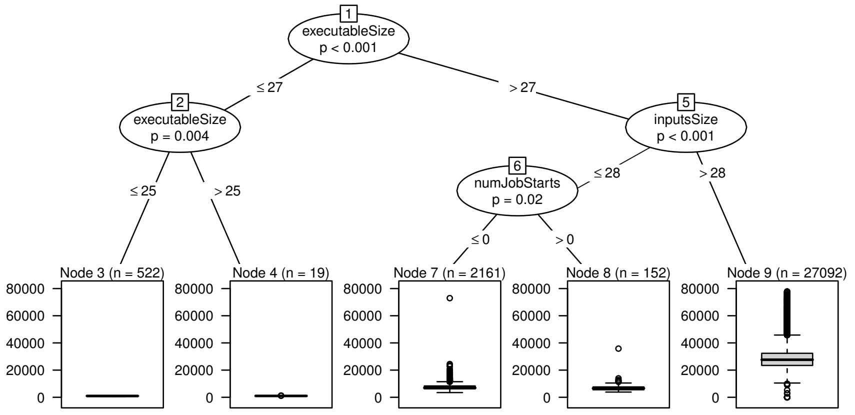


Figure 4: Example of a regression tree for one user of the dataset. Box plots show the distribution of job runtimes.

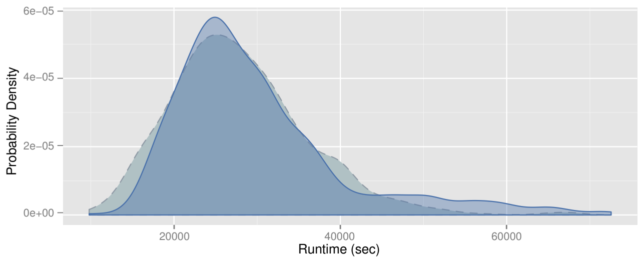


Figure 5: PDF of job runtime for one branch of the regression tree shown in Figure 4.

estimates (i.e., runtime, disk usage, and memory). Our recursive algorithm stops if the PDF of the subset fits a family of density, or if the variance is zero (e.g., job runtimes are constant). We conduct the Kolmogorov-Smirnov goodness of fit test (K-S test) [3] to determine whether a distribution of the subsets fits a probability distribution. In this work, we conduct K-S tests where the null hypotheses H_0 are defined as the data subset fits a Normal or a Gamma distribution. The two-sample KS test results in a miss if the null hypothesis is rejected at 5% significance level (p-value ≤ 0.05). For each probability function that the null hypothesis is not rejected (p-value > 0.05), we can then use the capabilities of these distributions to conduct further analysis. For instance, Figure 5 shows the PDF of job runtimes for one branch of the regression tree shown in Figure 4 (`executableSize` > 27 and `inputsSize` > 28). The PDF for the tree node (in blue) fits a Gamma distribution (grey dashed line) defined by a shape parameter $\alpha = 12$, a rate parameter $\beta = 5 \cdot 10^{-4}$, and mean $\mu = 27,414.8$. The distribution has a p-value $= 0.17$ for the K-S test, which does not reject the null hypothesis.

3 Job Estimation Process

The job estimation process is based on the analysis of the regression tree built for each user and estimation parameter. Terminal nodes of the regression trees are expressed as rules. The rule predicate is defined by the internal nodes (features) and a possible value of the feature. The rule output is defined by the mean, standard deviation, the number of jobs within the subset, and the result of the K-S test: whether the subset's PDF fits a Normal or a Gamma distribution, or none of them. For subsets that do not fit a distribution, values are generated according to a Uniform distribution. Figure 6 shows the overview of this process. Job characteristics such as the size of the application program and the size of the input data, are processed by a rule-based matcher to extract distribution parameters that best fits the job characteristics.

The estimation process uses the Marsaglia's polar method [12] to generate values from a Normal distribution. The method transforms from a two-dimensional continuous uniform distribution to a two-dimensional bivariate normal distribution. Suppose u and v are uniformly and independently distributed in $[-1, +1]$, then x and y have a normal distribution with mean

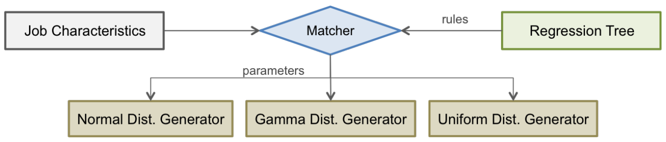


Figure 6: Overview of the job estimation process.

$\mu = 0$ and variance $\sigma^2 = 1$:

$$x = u\sqrt{\frac{-2\ln s}{s}}, \qquad y = v\sqrt{\frac{-2\ln s}{s}} \tag{2}$$

where $s = u^2 + v^2$. The method chooses random points $(u,\, v)$ until $s < 1$. In this work, the parameter μ (mean or expectation of the distribution) is defined as the median value of the PDF. Therefore, we approximate the generated values (Equation 2) from the normal distribution to $x' = \mu + \sigma \cdot x$, and $y' = \mu + \sigma \cdot y$, where σ is the standard deviation of the probability function. Note that the method generates a pair of independent standard normal random variables. When the estimation method asks for an estimate, x' is returned and y' is kept as a spare value for the next invocation.

For generating values from a Gamma distribution, the process uses the Marsaglia and Tsang method [13], since the shape parameter $\alpha \geq 1$ for fitted Gamma distributions in our dataset. Methods for generating values from a standard Normal distribution and a Uniform distribution are assumed to be available. Then, a gamma variate w can be generated as $d \cdot v$, defined as:

$$d = \alpha - \tfrac{1}{3}, \qquad v = \left(1 + \frac{x}{\sqrt{9 \cdot d}}\right)^3 \tag{3}$$

where x is a generated value from a Normal distribution. Note that this method generates Gamma variates for a scale parameter $\theta = 1$. Therefore, we approximate w to:

$$w' = \theta' \cdot d \cdot v, \qquad \theta' = \frac{1}{\beta} \tag{4}$$

where β is the rate parameter. Random variables generated from Uniform distributions are upper and lower bounded by the standard deviation distance from the mean.

4 Experiments and Evaluation

The experiment presented hereafter aims at evaluating the accuracy of the estimation process on predicting job requirements for the CMS workload such as runtime, disk, and memory usage.

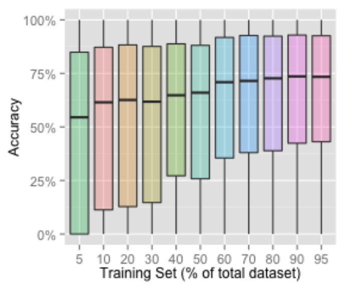
4.1 Experiment Conditions

Trace analyzes were performed on the dataset to build regression trees for each user of the CMS workload. To validate the estimation process, we built 11 distinguished sets of regression trees differentiated by the size of the training set, which ranges from 5% to 95% of the total dataset.

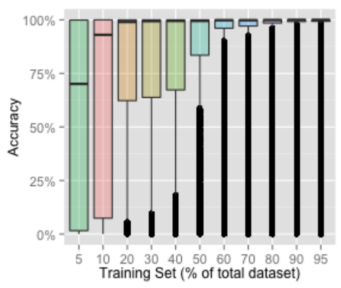
We implemented a simple job analyzer that receives job characteristics and the set of regression trees (in the form of rules), and outputs random values generated from a Normal, a Gamma, or a Uniform distribution. Regression tree sets derived from small training sets may not have rules for all users of the workload. In this case, random values are generated from a Uniform distribution, where the upper and lower bounds are defined by the maximum and minimum values respectively, of the estimated parameter in the training set. We do not aim to evaluate the efficiency of the estimation process algorithm or the process of building regression trees, but the accuracy of our process. Here, accuracy is defined as $1 - \text{abs}(\textit{actual} - \textit{predicted})/\textit{predicted}$.

4.2 Results and Discussion

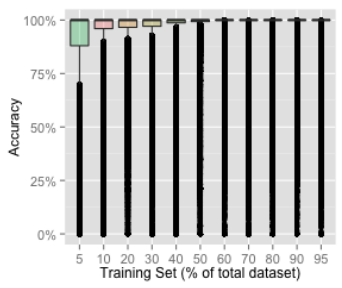
Figure 7 shows the average accuracy of the estimated parameters for the workload dataset, and Table 2 shows the summary of the number of rules per distribution for each parameter. Job



(a) Job Runtime



(b) Disk Usage



(c) Memory Usage

Figure 7: Average accuracy of the workload dataset for job runtime, disk usage, and memory usage estimation. The training set is defined as a portion of the entire workload dataset.

Training Set		Runtime				Disk Usage				Memory Usage			
	# Jobs	# Rules	Normal	Gamma	Uniform	# Rules	Normal	Gamma	Uniform	# Rules	Normal	Gamma	Uniform
5%	39,415	122	2	8	112	147	32	0	115	129	57	0	72
10%	78,831	205	46	35	124	206	42	1	163	180	98	1	81
20%	157,662	329	55	76	198	419	178	1	240	323	186	1	136
30%	236,493	404	107	81	216	536	192	1	343	409	269	1	139
40%	315,324	452	108	127	217	598	200	1	297	464	288	1	175
50%	394,155	520	109	143	268	678	251	1	326	529	296	1	232
60%	472,986	614	106	246	262	842	319	1	422	622	297	1	324
70%	551,817	641	104	250	287	936	333	1	602	668	293	2	373
80%	630,648	743	109	347	287	1064	354	1	709	761	301	2	458
90%	709,479	865	110	448	307	1174	359	2	813	844	322	2	520
95%	748,894	897	114	455	328	1213	364	1	848	863	335	2	526

Table 2: Number of rules per distribution for job runtime, disk usage, and memory usage.

runtime estimation (Figure 7.a) requires substantial data to provide fairly accurate estimates. This requisite is due to the high variability of job runtimes as shown in Table 1 (the standard deviation is within 160% of the mean value). For small training sets, most of the estimates are generated from Uniform distributions, which results in median accuracy values as low as 55%. Better accuracy is achieved when there is enough data to determine that the majority of the subsets (leaves of the regression trees) fits a Gamma distribution. A small training set of about 30% of the total dataset is enough to determine the subsets that fit a Normal distribution, but it is not sufficient to attain estimates with good accuracy.

Disk usage estimation (Figure 7.b) requires much less information for performing nearly optimal estimations. In contrast to the previous result, disk usage subsets fit mainly Normal distributions. For small training sets (10% or less of the total dataset), the median estimation accuracy is above 70%, but the first quartile yields accuracy values as low as 5%, which means the accuracy of the estimates for most of the dataset is low. For larger training sets, the number of rules indicating Normal distributions increases, and the estimation becomes more accurate. However, for very larger sets ($\geq$ 70%), there is significant increase in the number of rules directly proportional to the number of suggested Uniform distributions. This burst is due to the specialization of the partitioning algorithm, which tries to minimize the standard deviation. Few outliers (O(100)) that are not within the 1.5 IQR (interquartile range) have low accuracy. We conducted a separate analysis on these jobs to determine patterns that could distinguish them from the other jobs, but no explicit pattern could be found. This result suggests that the data collection process should be refined to gather finer information, or applications should provide mechanisms to distinguish custom user codes from the standard executable.

Similarly, memory usage estimation (Figure 7.c) is mostly based on Normal distributions. In fact, regression tree leaves that fit Normal distributions represent chunks of the PDF shown in

Figure 3 (right). For the smallest training set (5% of the total dataset), our estimation process results in estimates with 85% of accuracy for the first quartile. Nearly optimal estimates are achieved by using 40% of the total dataset as the training set. For larger training sets, the improvement in accuracy is negligible and there is a significant increase in the number of rules that indicate Uniform distributions (for the same reasons of the disk usage estimation). We also conducted a separate analysis of the outlier data, but no pattern could be found.

To measure the accuracy of our process in estimating future workloads, we also collected the workload of the HTCondor pool for the CMS experiment for the month of October 2014. The dataset consists of 1,638,803 jobs, where 810,567 are completed jobs submitted by 408 users. We then used the rules generated for the previous dataset (August 2014) to estimate job characteristics for the new dataset. The process results in a median estimation accuracy of 82% for job runtime, with 50% and 94% of accuracy for the first and third quartiles respectively. Not surprisingly, nearly optimal estimates (over 98%) are attained for disk and memory consumption. All the datasets and properties descriptions are available as part of the experimental material included in the Research Object (RO) associated with this work [15].

5 Related Work

Several methods for predicting job runtimes were proposed, but their application to real systems is often impracticable or inaccurate. Fei et al. [8] proposed a multicluster and multicloud resource management for resource provisioning. Their approach uses an auxiliary predictor module that estimates job runtimes as the mean of the runtimes of the last two finished jobs submitted by the same user. Sonmez et al. [17] studied job runtime and queue wait time prediction methods and their application in grid scheduling. They evaluated time series prediction methods when predicting job runtimes, and point-valued and upper-bound predictions when estimating queue wait times. A comparison to scheduling techniques that do not use prediction show that the use of these techniques does not imply better performance of the grid scheduling. Khan et al. [10] proposed a method to characterize and predict workloads in a cloud environment. Their method discovers and leverages repeatable workload patterns within groups of virtual machines (VMs) that belong to a cloud customer. A method based on Hidden Markov Modeling is used to capture temporal correlations and to predict the changes of workload pattern. The use of Markov-based techniques provides satisfactory accuracy for workload prediction, but it adds a significant overhead to the application execution.

6 Conclusions

We presented a simple, yet practical, method to estimate job characteristics such as runtime, disk usage, and memory consumption for the CMS workload. Our process is based on the analysis of regression trees built as a result of the use of statistical recursive partitioning method for each user and estimation parameter. Regression trees are expressed as rules, which define whether a subset of the dataset fits a probability distribution. The method was evaluated through the analysis of the dataset where the accuracy of our process was measured in comparison with real values. We also measured the accuracy of our process to estimate future workloads from historical data. Experimental results shows that our estimation process results in adequate estimates for job runtimes, and nearly optimal estimates for disk and memory consumption. Future work includes the analysis of the size of the historical data window used to build the rules, the analysis of techniques for limiting the specialization of the partitioning algorithm, and the analysis of preempted and failed jobs. We will also investigate other disk usage collecting mechanisms to better reflect the application's disk usage over time.

Acknowledgements

This work was funded by DOE under the contract number ER26110, "dV/dt - Accelerating the Rate of Progress Towards Extreme Scale Collaborative Science". We also thank Jeff Dost and Greg Thain for their valuable help, and the Open Science Grid (OSG).

References

[1] Stefani Belforte and et al. Using ssh as portal – the cms crab over glideinwms experience. *Journal of Physics: Conference Series*, 513(3):032006, 2014.

[2] Dan Bradley and et al. Use of glide-ins in CMS for production and analysis. *Journal of Physics: Conference Series*, 219(7):072013, 2010.

[3] Indra M. Chakravarty and et al. *Handbook of methods of applied statistics*. McGraw-Hill, 1967.

[4] Compact muon solenoid experiment. `http://cms.web.cern.ch`.

[5] CMS job exit codes. `https://twiki.cern.ch/twiki/bin/view/CMSPublic/JobExitCodes`.

[6] Rafael Ferreira da Silva and et al. Toward fine-grained online task characteristics estimation in scientific workflows. In *8th Workshop on Workflows in Support of Large-Scale Science*, 2013.

[7] Ewa Deelman, Gideon Juve, Maciej Malawski, and Jarek Nabrzyski. Hosted science: Managing computational workflows in the cloud. *Parallel Processing Letters*, 23(02):1340004, 2013.

[8] Lipu Fei, Bogdan Ghit, Alexandru Iosup, and Dick Epema. Koala-c: A task allocator for integrated multicluster and multicloud environments. In *Cluster Computing*, pages 57–65, 2014.

[9] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006.

[10] A. Khan et al. Workload characterization and prediction in the cloud: A multiple time series approach. In *IEEE NOMS*, pages 1287–1294, 2012.

[11] Muthucumaru Maheswaran and et al. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *8th Heterogeneous Computing Workshop*, 1999.

[12] George Marsaglia and Thomas A Bray. A convenient method for generating normal variables. *Siam Review*, 6(3):260–264, 1964.

[13] George Marsaglia and Wai Wan Tsang. A simple method for generating gamma variables. *ACM Trans. Math. Softw.*, 26(3):363–372, 2000.

[14] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999.

[15] Research object. `http://pegasus.isi.edu/publications/cms`.

[16] Igor Sfiligoi. Estimating job runtime for CMS analysis jobs. *Journal of Physics: Conference Series*, 513(3):032087, 2014.

[17] Ozan Sonmez et al. Trace-based evaluation of job runtime and queue wait time predictions in grids. In *18th ACM HPDC*, pages 111–120. ACM, 2009.

[18] Sen Su and et al. Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing*, 39(4–5):177–188, 2013.

[19] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[20] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, 2002.

[21] J. D. Ullman. Np-complete scheduling problems. *J. Comput. Syst. Sci.*, 10(3):384–393, June 1975.

[22] Shichao Zhang, Chengqi Zhang, and Qiang Yang. Data preparation for data mining. *Applied Artificial Intelligence*, 17(5-6):375–381, 2003.