A Host-Independent Supervised Machine Learning Approach to Automated Overload Detection in Virtual Machine Workloads

Eli M. Dow
IBM Research
Yorktown, NY
emdow@us.ibm.com

Jeanna N. Matthews
Clarkson University
Potsdam, NY
jnm@clarkson.edu

Abstract—**This paper evaluates a mechanism for applying machine learning (ML) to identify over-constrained IaaS virtual machines (VMs). Herein, over-constrained VMs are defined as those who are not given sufficient system resources to meet their workload specific objective functions. To validate our approach, a variety of workload-specific benchmarks inspired by common Infrastructure-as-a-Service (IaaS) cloud workloads were used. Workloads were run while regularly sampling VM resource consumption features exposed by the hypervisor. Datasets were curated into nominal or over-constrained and used to train ML classifiers to determine VM over-constraint rules based on one-time workload analysis. Rules learned on one host are transferred with the VM to other host environments to determine portability. Key contributions of this work include: demonstrating which VM resource consumption metrics (features) prove most relevant to learned decision trees in this context, and a technique required to generalize this approach across hosts while limiting required up front training expenditure to a single VM and host. Other contributions include a rigorous explanation of the differences in learned rulesets as a function of feature sampling rates, and an analysis of the differences in learned rulesets as a function of workload variation. Feature correlation matrices and their corresponding generated rule sets demonstrate individual features comprising rule sets tend to show low cross-correlation (below 0.4) while no individual feature shows high direct correlation with classification. Our system achieves workload-specific error percentages below 2.4% with a mean error across workloads of 1.43% (and strong false negative bias) for a variety of synthetic, representative, cloud workloads tested.**

Keywords—virtual machines; cloud-computing; IaaS management; decision trees; support vector machines; binary classification; cloud provisioning; resource allocation.

I. Introduction

Generally, existing systems require the VM operator to install a software agent in the VM that can be used to gather information and drive a monitor-analyze-remediate management loop for VM resource allocation decisions. Out-of-band monitoring, or agentless monitoring, is simpler to install and administer (centralized on the hypervisor not the VMs) and does not restrict the VM owner by imposing an agent or any other requirements into their VM. The lack of interference or imposed requirements on IaaS cloud VMs cannot be understated, as this cloud type caters to arbitrary OS images.

While out-of-band agents suffer from reduced visibility into the behavior of the system being monitored, the benefits of black-box monitoring for VM management in IaaS makes it a compelling research prospect. The long running (indefinite execution) nature of VM processes indicate that the overhead associated with the measurement and detection needed for load balancing may be well worth the cost in contrast to short lived process management where black box techniques are typically not used. This work aims to improve existing systems through the more challenging, and more applicable to the IaaS context, black-box monitoring approach in an host-independent context.

II. Related Work

A variety of prior work seeks to determine over-constraint/overload detection in real time, or by anticipating future workload demands. However, these previous efforts select, *a priori*, the feature of interest to trigger their remediation on based on intuition. They most often use CPU consumption (or another univariate trigger). Beyond static thresholding, some recent work uses heuristics based on statistical analysis of historical VM resource consumption [1,2], effectively using adaptive thresholds based on statistical properties [3]. One alternative technique uses pattern-driven application consolidation by examining patterns of resource usage (signatures in their work), and applying dynamic time warping and the fast Fourier transform to seek a match between available host capacity and individual VM resource demand signatures [4]. Though that effort was principally aimed at periodic global consolidation in environments with high workload periodicity, instantaneous overload/over-consolidation strategies were discussed, relying on a single predictor exceeding a static threshold. In contrast to these efforts, we use a data driven approach to determine a multivariate over-constraint indicator.

Other recent works focus on the shift from immediate overload detection to anticipatory systems by applying a variety of techniques such as: linear regression [5] for short term CPU prediction, auto-regressive integrated moving averages [6], and more sophisticated schemes that apply longer term prediction augmented with uncertainty estimation [7] (though that work also defines a key concept of SLA violation in terms of CPU consumption of the VM compared to allocated CPU share for that VM). Our work differs through use ML to define the salient characteristics indicating over-constraint. Our technique can employ any of these forecasting methods to move into an anticipatory modality, and thus may enhance these systems by projecting recent feature observations into the future (using the preferred forecasting technique) and inputting those projections into the machine learned, multivariate, over constraint classifier.

Our work is most strongly inspired by Vigilant, a system based on out-of-band monitoring. Vigilant determined the health and status of a VM based solely on data observations that were accessible from the hypervisor context [8]. Vigilant specifically targeted the detection of extremely high CPU utilization in kernel space. The experimental results from Vigilant show that certain types of problem conditions in VMs could be detected out-of-band with high accuracy while avoiding the pitfalls associated with in-band monitoring. Like Vigilant, our system monitors the hypervisor for VM utilization information and uses a decision tree classifier ML method [9] to analyze the readings at run time and detect problems *in situ*. Though Vigilant was the first system to combine the ease of out-of-band data collection with ML to improve VM systems management, their work was focused on detecting a limited set of faults (e.g., halt on error conditions). Our own work provides a generalization of their approach to autonomously detecting VM over-consolidation. Those curious about the details of the virtualization differences in Vigilant and our own work should note their work was based on the QEMU emulator [11] with each emulated machine running a different type of workload (web service, mail service, etc.). Since QEMU underpins the operation of KVM virtual machines [12] it follows then that their preliminary results should guide our investigation using KVM.

III. Research Questions and Methodology

Motivated by interest into autonomous over-constraint detection for VMs, this work asks the following research questions:

1. *Which features or resource metrics are most useful in automatically identifying over-constraint in IaaS cloud computing environments?*

2. *How do the rule sets of learned classifiers vary across common IaaS VM workload types?*

3. *How portable are learned classifiers? Specifically, can they be implemented such that an unmodified classifier can travel with the virtual machine to other hosts while retaining applicability?*

4. *What are the differences in learned classifiers as a function of sampling frequency? Can one determine a lower bound on practical sampling intervals?*

Experimentation was done on a KVM hypervisor using a VM created from scratch and configured with a variety of middleware and workloads to be executed while under observation from our black-box VM monitoring software (running on the hypervisor). Both the hypervisor OS and VM were Ubuntu version 16.04 (the current release at the time of experimentation). The workloads installed on the VM are enumerated in section *A: Workload Overview* while data collection methodology is explained in section *B: Experimental Data Recording*. Each workload was run (while data was collected from within the VM to annotate over-constraint conditions) concurrently with externally visible data collection at the hypervisor. Workload-aware data (for ground truth) enables supervised ML on the black box data.

A. Workload Overview

A series of representative, synthetic, surrogate IaaS workloads were identified as a basis for evaluation spanning three major categories:

1) HTTP-PHP Workload: Workloads are executed against our HTTP [13] and PHP [14] installation by using the *apachebench* benchmark utility[1] [15]. The individual runs were configured to flag unconstrained VM situations when 90% of page responses were received in less than 3 seconds. When more than 10 percent of the page responses take more than 10 seconds, we consider the HTTP VM server to be over-constrained. This threshold was based on the observation that web pages that fail to load instantly are considered too slow for production. Consumers invariably become frustrated with latencies, especially on mobile devices. Individual page responses taking longer than 5 seconds to return are tallied as errors. The number of concurrent requests were scaled iteratively from 10, 20, 30, 40, 50, 100, 150, 200, 250, 300, 350, 400, 500, and 1000. Each iteration is run for 60 seconds before the next iteration to allow a gradual workload ramp up. The workload logs the start and end time of each iteration as well as when the over-constraint event.

2) Database Workload: To execute a workload against the database instance, we installed the *mysqlslap*[2] [16]. Overload was defined by any individual slap invocation with an average query latency exceeding 5 seconds. The workload similarly logs start, end, and over-constraint event timing.

3) I/O Dominated Workloads (Simulated File Serving): To simulate file I/O oriented workloads performed in the cloud the *filebench* workload engine was selected and configured to emulate the file access patterns of a streaming video server. This configuration simulated new content creation (uploads) and aging out of older content in such a way as to simulate services like YouTube. The workload has 2 components, one which creates new content and one serving the content. The workload distinguishes between actively served videos and aged content (paged out to disk). We manipulated the number of threads representing streaming users over the range of [40-60], incrementing by one additional thread per iteration until the number of reported I/O operations per second fell below a configured threshold (300). Each iteration ran for at least 60 seconds, with videos replaced once every 10 seconds.

B. Experimental Data Recording

To record a pure data set, each VM was exercised in isolation on an idle KVM host. Our black-box monitoring software was started early to capture the full VM boot up process as well as a training workload interval and an idle interval. Since patterns of resource consumption at boot-up are often drastically different than operational workloads, VM restarts under monitoring might otherwise appear to show atypical workload profiles from out-of-band. The VMs reached a quiescent state of activity prior to starting workloads, and were run until over-constraint was triggered. Values logged during black-box hypervisor data collection are obtained principally from the */proc* virtual file system [17] on the hypervisor.

Samples were obtained at 1Hz for each of the features monitored and are listed with explanation in Table I. Fig. 1 illustrates a plot that is representative of the fundamental data collection for each of the experiments performed. The figure, representative of the data collected from all experiments, was taken from a single HTTP run. The abscissa shows time in seconds from experiment start, covering approximately 10 minutes in elapsed time. Each stacked subplot shows the time

[1] To test our HTTP-based workload we performed a nominal installation of the Apache web server using the "apt-get install apache2" command. Once a minor change was made to the default configuration of Apache to silence a configuration-file related warning, we proceeded to install PHP to create more realistic web content to serve based on the observation that web pages are not simply static HTML. Once installed and configured, a simple PHP page was created to exercises the PHP runtime.

[2] A freely available database linked from the MySQL documentation page (dev.mysql.com/doc/index-other.html) was installed to provide reasonable test data for the MySQL workload. The database is described as "employee data (large dataset, includes data and test/verification suite)." Since one table had almost half a million rows of populated data, the design criteria for a sufficiently large dataset was satisfied.

series of observations from a single feature. Only features with nonzero series are shown (some features are not shown where no data was collected). Several seemingly relevant features hypothesized to be useful in Table 1 produced no usable observations in our experiments. Those features showing predictive power are marked in bold. Upon investigation, many parameters are not relevant to KVM VMs (though would likely be relevant in other hypervisors). Several constituent feature time series show high correlation but are in fact distinct when viewed at high resolution. The left most 1/6th of Fig. 1 corresponds to VM boot-up. Series labels in inset boxes correspond to the entries in Table I. Post-processing of instrumented workload logs created during the collection of this data allowed annotation as nominal or over-constrained, forming the basis of our supervised binary classification scheme.

IV. Experimental results

This section provides an analysis of our experimental results. First, we present an overview of the multivariate data we collected across workloads and our analysis approach. Second, this section describes the features that were found to initially have predictive power from our set of reasonable guess features in Table II. Following that, we present an investigation into the effect of feature observation variance on learned rules, quantifying rule set accuracy. This section concludes with an analysis of networking setup and observation sampling regime sensitivity as well as analysis of learned rule set portability.

For each workload run, feature-specific data was analyzed using a series of automated plots and analysis for manual inspection. This analysis included time series plots of each feature to observe general trends in the series as well as ensuring each observation was recorded correctly. Box and whisker plots were created for each feature to characterize the variance of feature observations. Histograms of observed features were created as well as a scatterplot of each feature against the determined classification. The histogram plots of observed values were performed to determine what, if any, type of distribution the feature corresponds to. Many ML models assume a Gaussian distribution, and surprisingly none of our observations corresponded to a Gaussian distribution. In the scatterplot analysis of each observation, the corresponding determined classification was encoded as a 1 or 0, with 1 meaning over-constrained or under-performing the experimental threshold for successful virtual machine operation response, and 0 meaning nominal or well-performing VM response.

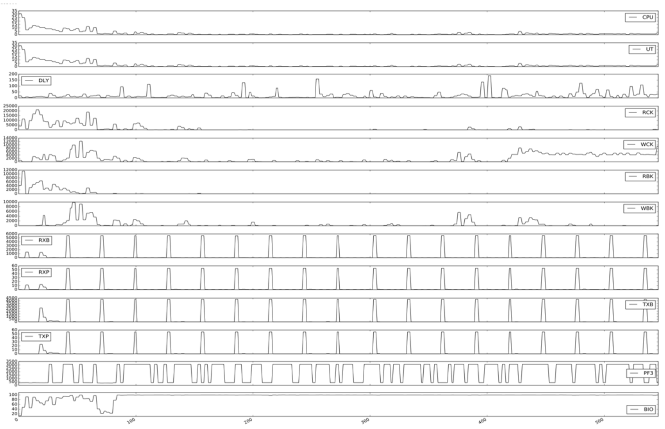

Fig. 1. An example experiment time series set showing observed feature series shown to illustrate high-level feature series similarity across some feature subplots. Such subplots, in higher resolution than can be reproduced here for publication length limitations, were used to validate our collection framework.

TABLE I. OBSERVED KERNEL FEATURES

Feature	Description
CPU	Amount of time that this process was scheduled in both kernel and user-blospace time as a percentage of all time elapsed. From /proc/[pid]/stat
UT	Amount of time that this process has been scheduled in user mode, measured in clock ticks. This includes guest time, guest_time (time spent running a virtual CPU, see below), so that applications that are not aware of the guest time field do not lose that time from their calculations. "Utime" value from /proc/[pid]/stat
ST	Amount of time that this process has been scheduled in kernel mode, measured in clock ticks. "Stime" value from /proc/[pid]/stat
CUT	Amount of time that this process's waited-for children have been scheduled in user mode, measured in clock ticks. This includes guest time, cguest_time (time spent running a virtual CPU, see below). "Cu_time" value from /proc/[pid]/stat
CST	Amount of time a process's waited-for children have been scheduled in kernel mode, in clock ticks. "Cs_time" value from /proc/[pid]/stat
GT	Time spent running a virtual CPU for a guest operating system, in clock ticks. "Guest_time" value from /proc/[pid]/stat
CGT	Guest time (GT) of the process's children, measured in clock ticks. "Cguest_time" value. from /proc/[pid]/stat
DLY	Obtained from /proc/pid/schedstat, indicates the time spent waiting on a kernel run queue but not executing.
RCK	The number of bytes which this task has caused to be read from storage. This is simply the sum of bytes which this process passed to read() and pread(). It includes things like tty IO and it is unaffected by whether actual physical disk IO was required (the read might have been satisfied from page cache). From /proc/[pid]/io
WCK	The number of bytes which this task has caused, or shall cause to be written to disk. Similar caveats apply here as with RCK. From /proc/[pid]/io
RBK	The number of bytes which this process fetched from the storage layer. Done at the submit_bio() level. From /proc/[pid]/io
WBK	Number of bytes which this process caused to be sent to the storage layer. This is done at page-dirtying time. From /proc/[pid]/io
RXB	Received bytes from virDomainInterfaceStats struct in libvirt
RXP	Received packets from virDomainInterfaceStats struct in libvirt
RXE	Receiver side errors from virDomainInterfaceStats struct in libvirt
RXD	Receiver side dropped packets from virDomainInterfaceStats struct in libvirt
TXB	Transmitted bytes from virDomainInterfaceStats struct in libvirt
TXP	Transmitted packets from virDomainInterfaceStats struct in libvirt
TXE	Transmission side errors from virDomainInterfaceStats struct in libvirt
TXD	Transmission side dropped packets from virDomainInterfaceStats struct in libvirt
PF1	Count of minor faults the process has made which have not required loading a memory page from disk. "Minflt" value from /proc/[pid]/stat.
PF2	The number of minor faults that the process's waited-for children have made. "Cminflt" value.
PF3	The number of major faults the process has made which have required loading a memory page from disk. "Majflt" value from /proc/[pid]/stat.
PF4	The number of minor faults that the process's waited-for children have made. "Cmajflt" value from /proc/[pid]/stat.
BIO	Aggregated block I/O delays, measured in clock ticks, expressed as a rate per time. "Delayacct_blkio_ticks" value from /proc/[pid]/stat.

In addition to univariate analysis of features, an examination of the multivariate data set (across all features) from each experiment was performed. A representative example Andrews plot [18] for the HTTP workload is shown in Fig. 2. A type of signal, shown in teal (darker) distinguishes over-constrained data values against background of gold (lighter) nominal values. This signal is what our ML approach attempts to discern from the feature vector observations. Since some signal seems apparent, we next sought to answer if any individual variable had overwhelming predictive power or if features showed high cross-correlation. Each attribute of an observational data set row is represented by a point on the line, like a line chart, but the way data is translated into a plot is substantially different. Each column from the data set is normalized independently and smoothed.

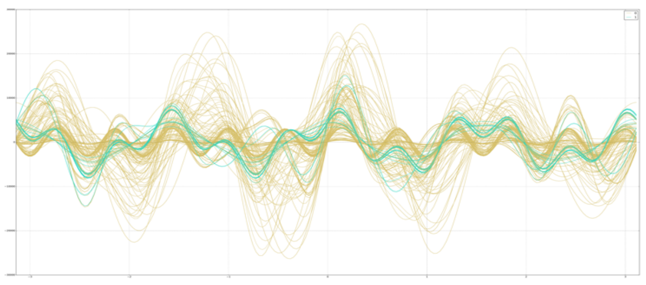
Fig. 2. One representative Andrews Plot from an HTTP workload as described in subsection: "HTTP-PHP Workload". ML seeks the signal shown.

Correlation matrix plots across all feature time series were created with the Pandas Python library to highlight positive and negative correlations in the observations. By focusing on intra-feature correlation, efficiency can be increased by sampling only one element from a highly-correlated feature set at runtime. Example correlation plots are shown in Fig. 3, 4, and 5. Entirely white cells represent features that did not yield experimental data during VM observation and thus cannot be correlated. The value 1.0 indicates perfect positive correlation, while -1.0 represents perfect negative correlation. Since some data are related (i.e., received bytes and packets) a degree of correlation is expected. We anticipated that elements from a highly-correlated feature pairs would not simultaneously occur in learned rulesets. By including the classification in the correlation matrix we determine if any individual feature is overwhelmingly predictive, implying an existence of a sufficiently viable univariate over-indicator. No individual feature was highly correlated to learned classification suggesting validity in our multivariate supervised ML approach.

A. Initial Features Found to Have Predictive Power

In the following section, Table II summarizes predictive features from our initial experimentation using 1Hz sampling across workloads. Marked values show predictive power.

Our intuition of the HTTP workload was that some combination of delay and networking features would dominate the rule set. The results were more nuanced than expected. Experimentally, rules were comprised of CPU consumption (CPU), blocked I/O (BIO) and memory writes (WCK and WBK). The lengthiest rule generated indicates over-constraint in a narrow range of CPU consumption when blocked I/O and CPU scheduler delay were high. Simpler rules indicate that blocked I/O and memory pressure below a certain threshold indicated an HTTP VM is likely to be nominal.

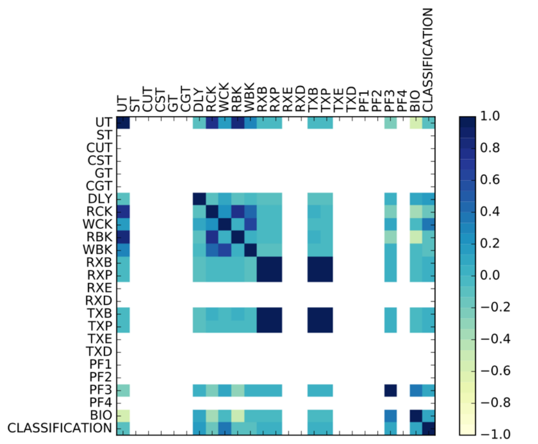

Fig. 3. Feature Correlation Matrix HTTP Workload.

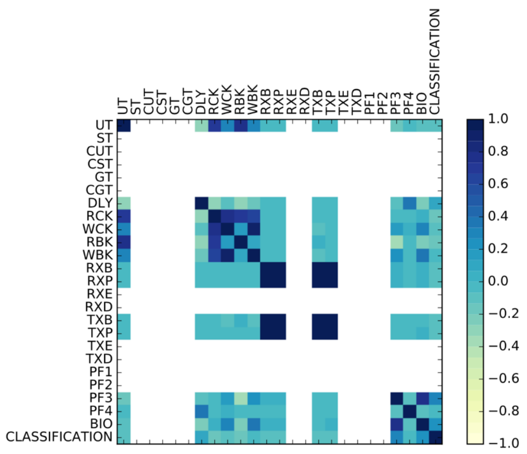

Fig. 4. Feature Correlation Matrix SQL Workload

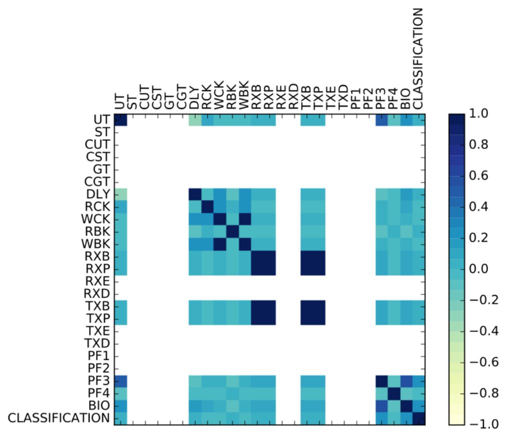

Fig. 5. Feature Correlation Matrix Video Workload

TABLE II. FEATURES USED IN FINAL DECISION TREE RULES

Kernel Feature	HTTP/PHP Web Server	SQL Database	Video Server (I/O)
CPU	•		•
DLY	•		
RCK	•	•	
WCK	•	•	•
RBK			•
WBK	•		
PF3			•
PF4			•
BIO	•	•	•

Our assumption for the SQL workload was that the rules would be generally governed by blocked I/O (BIO) and CPU properties. Our intuition did not entirely match reality. The rules were indeed governed by blocked I/O, but memory characteristics comprised the remainder of the interesting features. Two resulting rules indicted a machine learned tipping point for this workload in the WCK property, when in conjunction with high blocked I/O, causes a classification to transition from nominal to over-constrained.

Having no prior experience with our file streaming workload, we had no intuition. Interestingly, combinations of CPU consumption (CPU), a specific page fault type (PF3), and blocked I/O (BIO) make up most of the rules. Page faults may be related to requesting un-cached content on disk, or requesting aged-out content. Similarly, processes blocked on I/O seem a relevant indictor for streaming content workloads.

B. *Effect of Observation Variance on Learned Rules*

Many of the features sampled are far more variable than others. Experimentation was necessary to explore the relationship between feature variability and machine learned decision tree rules. One method for understanding these workload traces is to visualize the features sampled as a set of coordinates in a high-dimensional data space using 1 axis per variable. This technique is known as Principal Component Analysis (PCA)[3] [19]. Using this approach, a PCA plot constructs a lower-dimensional projection of the data when viewed from a particularly advantageous viewpoint highlighting variance. Each workload trace feature was mapped into the range [-1, 1] before plotting the PCA (lowest mode) to determine features with the largest variance. Analysis of PCA feature magnitude and learned rules indicate feature salience is unrelated to variance for this domain.

C. *Rule Set Accuracy*

In addition to generating rulesets, the classifier runtime provides measures of rule accuracy against training data reserved for testing at rule generation time [9]. Table III lists the rule set error and information about the observations used to generate the rule sets. The column labeled "Tested" indicates the number of data rows (time steps) used to train the classifier with nominal and over constrained data. All values are for a 1Hz sampling regime, and thus runtimes can be inferred for the experiments: ~10, 4, and 16 minutes respectively for the HTTP, SQL, and VIDEO workloads respectively. The "Error" column indicates cumulative false positives and negatives encountered during rule testing performed by the classifier at rule generation time. The "Error %" column indicates the percentage of classifications in error if the rule set had been in effect. Note the low error percentages with a distinct trend towards false negatives.

TABLE III. DECISION TREE ACCURACY

Workload	Tested	Errors	Error %	False Positive Count	False Negative Count	Precision	Recall
HTTP	547	6	1.1%	1	5	0.998	0.991
SQL	260	2	0.8%	1	1	0.996	0.996
VIDEO	1009	24	2.4%	0	24	1.000	0.976

D. *Sensitivity to Network Configuration*

Referring again to Table II, we expected more features from Table I to demonstrate predictive value (e.g., the lack of network related features from the HTTP case). In effect, the decision tree rules guided further research. To investigate intra-hypervisor vs. externally driven workload sensitivity, we ran the HTTP workload from an off-hypervisor host instead of a co-resident VM. Results are shown in Table IV. This change yielded predictive power in both a receipt and transmission feature (RXB and TXP) demonstrating predictive features and rulesets are sensitive to network configuration at training time. Note the indication of PF3 as predictive for this variant while it was not selected in the intra-hypervisor experimental HTTP rule set. Error rates were small in both networking experiments and generally consistent. More expected features were present in the off-platform client scenario.

TABLE IV. EXTERNAL VS. INTERNAL NETWORKING EQUIPMENT

HTTP Feature	1HZ HTTP INTERNAL NETWORKING	1HZ HTTP EXTERNAL NETWORKING
CPU	•	•
UT		•
DLY	•	•
RCK	•	•
WCK	•	•
WBK	•	•
RXB		•
TXP		•
PF3		•
BIO	•	•
ERROR %	1.1% (6/547)	1.4% (4/287)

E. *Sensitivity to Observation Sampling Frequency Variation*

Classifiers were trained with subsampled data (3, 5, 10, and 60 second windows to determine the effects of feature sampling frequency regime on rules. Results are shown in Table V. Note that no classifier generated a viable rule set at the 1 Minute sampling interval, wherein default classifications were always assumed. Thus, those columns were omitted entirely, as was the case in the SQL workload sampled at 1/10 Hz. There are two potential explanations. The first assumes that the data series is too sparse, and through sufficient experimental replication one could build a viable classifier from a longer training series. A second explanation is that the fundamental patterns in the data are on time scales much faster than once per minute and that transient events indicative of over-constraint get lost in such regimes. Varying observation sampling frequencies yields a quantifiable impact on rulesets (including accuracy against training tests as well as feature predictive power). Decreasing sampling rates yield rule sets based on fewer features. The most commonly predictive feature was blocked I/O. Since each workload had an I/O component this is not entirely surprising.

[3] PCA is also known by other names depending on the field of application including but not limited to: the discrete Kosambi-Karhunen–Loève transform in signal processing, the Hotelling transform in multivariate quality control, proper orthogonal decomposition in mechanical engineering, and eigenvalue decomposition in linear algebra.

TABLE V. CLASISFIER SENSITIVITY TO VARYING THE FEATURE COLLECTION INTERVAL

Feature	HTTP				SQL			Video Streaming			
Freq. (secs)	1	3	5	10	1	3	5	1	3	5	10
CPU	●							●		●	
DLY	●										
RCK	●				●	●	●				
WCK	●	●	●	●	●			●			
RBK				●				●	●		
WBK	●	●	●								
RXB									●		
RXP						●				●	
TXP			●								●
PF3			●					●			
PF4								●			
BIO	●				●	●	●	●	●	●	●
Error %	1.1	6	1.8	1.8	0.8	0	1.9	2.41	3.3	2.0	2.0
Error Count	6	11	1	2	2	0	1	24	11	4	2
Tests	547	183	55	110	260	87	53	1009	337	292	101

F. Rule Set Portability

To test classifier portability, we trained on a small system[4], then moved the VMs and trained classifiers to a larger system[5] prior to re-running the workload. Large host observations were used with the classifier generated on the small host using the same 1Hz sampling regime. Relocated classifiers worked across hosts in each of our workloads, due in part to the design of the classifier input data. Training data features are not expressed as exposed natively but rather converted to rates per sampling interval. Other features were expressed as a percentage of absolute physical capacity. This allows training and classification with host-abstracted data for more generalizable results. However, we assume a practical limit to generalizability across hardware (e.g., solid state disk vs. rotational media). While experimentation across a broad variety of hardware is required to definitively explore these limits, IaaS often leverages generally homogeneous hardware [20]. Our results indicate viability in practice when using comparable hardware. Horizontal scaling (cloned VMs) should work well, with each VM autonomously and independently managed for overload wherein the triggered remedial action may involve orchestration software for elasticity or workload balancer adjustment. Portability can be improved as outlined in our previous work [21] to compile rules into embeddable, shared objects on a per-VM basis as specified in a VM contract [22].

IV. Conclusions And Future Work

We observed low error rates in automated detection of over-constrained VM's using decision tree classifiers trained on hypervisor-exposed features, and showed ruleset portability. We showed the impact of temporal regime on classifier rules. In summary, the inapplicability of 1 minute sampling intervals and negligible overhead at 1Hz sampling imply that scale-out to hundreds of VMs should be reasonable for triggering timely remediation. We highlighted the importance of constructing realistic training with respect to VM networking. By analysis of feature correlation matrices and rulesets, we note low feature cross correlation (below 0.4). While we demonstrated that no individual feature has high direct correlation with over-constraint (suggesting potentially invalid assumptions in related work), a focused selection on a workload-specific combination of features can be very predictive. We note that machine learned predictive feature sets were relatively small in comparison to the full list hypothesized to be relevant. Rulesets were generally short (6-9 rules), with each consisting of at most 5 expressions. Lastly, our approach is not tied to VMs since our data are derived from */proc* entries and may apply to other workloads encapsulated as individual processes, thus extension of our work along those lines may be related in spirit to other research [23].

[4] The small system was a 4-core i5 processor at 2.67 GHz with 3GB RAM.

[5] The large system was a 24-core Xeon ES02540 at 2.5GHz with 94GB RAM

References

[1] A. Beloglazov *et al*., "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing" in *Future Generation Comp Systems* 28(5): 755–768. 2012.

[2] R.N. Calheiros et al., "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms" in *Software: Practice & Experience* 41(1): 23–50. 2012

[3] T.C. Ferreto *et al*., "Server consolidation with migration control for virtualized data centers" in *Future Generation Comp Syst* 27(8): 1027–1034. Elsevier B.V., Amsterdam.

[4] Z. Gong, X. Gu. "Pac: Pattern-driven application consolidation for efficient cloud computing" in *Proc 2010 IEEE Int. Symp. on Modeling, Analysis and Sim. of Computer and Telecom. Sys.*, 24–33.. IEEE Computer Society, Washington.

[5] F. Farahnakia *et al*., "Lircup: Linear regression based CPU usage prediction algorithm for live migration of virtual machines in data centers" in *Proc. of the 2013 39th Euromicro Conf. on Software Eng & Advanced Applications*, 357–364. IEEE. Computer Society, WA.

[6] S. Khatua *et al*., "Prediction-based instant resource provisioning for cloud applications" in *Proc. of the 2014 IEEE/ACM 7th Int. Conf. on Utility and Cloud Computing*, IEEE Computer Society, Washington. pp 597–602.

[7] D. Minarolli *et al*. "Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing" in *Journal of Cloud Comp Cloud Comp* (2017) 6: 4.

[8] D. Pelleg *et al*., "Vigilant: out-of-band detection of failures in virtual machines" in *SIGOPS Operating. Systems. Review.* 42, 1 (Jan 2008)

[9] J. R. Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[10] R. Duda *et al*., *Pattern Classification*. Wiley. ISBN: 978-0-471-05669-0

[11] F. Bellard. Qemu, a fast and portable dynamic translator in *USENIX 2005 Annual Technical Conf., FREENIX Track*, pages 41–46,.

[12] A. Kivity. KVM: the Linux Virtual Machine Monitor in *OLS '07: The 2007 Ottawa Linux Symp.*, pages 225–230, July 2007.

[13] Apache HTTP Server Project. Available: http://httpd.apache.org/

[14] PHP website. Accessed January 2017 at http://www.php.net/.

[15] Apache HTTP Server Benchmarking Tool User Manual- Available: http://httpd.apache.org/docs/2.4/en/programs/ab.html

[16] MySQL-SLAP Reference from the MySQL Database Project. Available: https://dev.mysql.com/doc/refman/8.0/en/mysqlslap.html

[17] T. J. Killian, "Processes as Files," in *USENIX Summer Conf. Proc.*, Salt Lake City, UT, USA (June 1984).

[18] C. García-Osorio, and C. Fyfe. "Visualization of High-Dimensional Data via Orthogonal Curves" in *Journal of Universal Computer Science.* 11 (11): 1806–1819. 2005

[19] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space" in *Philosophical Magazine.* 2 (11): 559–572.

[20] L. A. Barroso and U. Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines (1st ed.).* Morgan and Claypool Publishers. 2009. Available: http://bnrg.eecs.berkeley.edu/~randy/Courses/CS294.F09/wharehousesizedcomputers.pdf

[21] E. Dow, and T. Penderghest. "Transplanting Binary Decision Trees" in *Journal of Computer Sciences and Applications* 3.3 (2015): 61-66.

[22] J. Matthews *et.al.*, "Virtual machine contracts for datacenter and cloud computing environments" in *Proc. of the 1st Workshop on Automated Control for Datacenters and Clouds*, 2009, pp. 25-30

[23] S. Kundu *et al*., "Modeling virtualized applications using machine learning techniques" in ACM SIGPLAN Notices. Vol. 47. No. 7. ACM. 2012