# Resource Monitoring and Management with OVIS to Enable HPC in Cloud Computing Environments

Jim Brandt*, Ann Gentile°, Jackson Mayo*, Philippe Pébay*,
Diana Roe°, David Thompson*, and Matthew Wong°
Sandia National Laboratories
MS *9159 / °9152
P.O. Box 969, Livermore, CA 94551 U.S.A.
{brandt,gentile,jmayo,pppebay,dcroe,dcthomp,mhwong}@sandia.gov

***Abstract*—Using the cloud computing paradigm, a host of companies promise to make huge compute resources available to users on a pay-as-you-go basis. These resources can be configured on the fly to provide the hardware and operating system of choice to the customer on a large scale. While the current target market for these resources in the commercial space is web development/hosting, this model has the lure of savings of ownership, operation, and maintenance costs, and thus sounds like an attractive solution for people who currently invest millions to hundreds of millions of dollars annually on High Performance Computing (HPC) platforms in order to support large-scale scientific simulation codes. Given the current interconnect bandwidth and topologies utilized in these commercial offerings, however, the only current viable market in HPC would be small-memory-footprint embarrassingly parallel or loosely coupled applications, which inherently require little to no inter-processor communication. While providing the appropriate resources (bandwidth, latency, memory, etc.) for the HPC community would increase the potential to enable HPC in cloud environments, this would not address the need for scalability and reliability, crucial to HPC applications. Providing for these needs is particularly difficult in commercial cloud offerings where the number of virtual resources can far outstrip the number of physical resources, the resources are shared among many users, and the resources may be heterogeneous. Advanced resource monitoring, analysis, and configuration tools can help address these issues, since they bring the ability to dynamically provide and respond to information about the platform and application state and would enable more appropriate, efficient, and flexible use of the resources key to enabling HPC. Additionally such tools could be of benefit to non-HPC cloud providers, users, and applications by providing more efficient resource utilization in general.**

## I. Introduction

### A. Cloud Computing and HPC

Using the cloud computing paradigm [1], a host of companies promise to make huge compute resources available to users on a pay-as-you-go basis. These resources can be configured on the fly to provide the hardware and operating system of choice to the customer on a large scale. A customer who has finished simply releases the resources back into the pool from which the next customer draws. Customers avoid the up-front cost of designing, procuring, and setting up a system, which can be considerable even with respect to just power and cooling systems and facilities. There is no cost of ownership and no cost of disposal. While the current target market for these resources in the commercial space is largely web development/hosting, this model sounds like an attractive solution for people who currently invest millions to hundreds of millions of dollars annually on HPC platforms in order to support large-scale scientific simulation codes.

Given the interconnect speeds and topologies utilized in current commercial offerings, however, the only viable market in HPC would be small-memory-footprint embarrassingly parallel or loosely coupled applications, which inherently require little to no inter-processor communication. However, expanding the applicability of commercial cloud platforms to HPC applications will require more than just providing the right resources (bandwidth, latency, memory, etc.).

The needs for scalability and reliability manifest themselves in HPC applications differently from web or serial software applications. Current scientific simulation codes typically rely on MPI as the underlying communication protocol, with Infiniband or Ten Gigabit Ethernet as the physical transport medium between processors not residing on the same motherboard, and shared memory as the physical transport between processors that do reside on the same motherboard. Though there are fault-tolerance mechanisms being integrated into some MPI implementations, this is currently an area of vulnerability for many of these codes. When some piece of hardware or software fails, causing failure of one or more MPI processes, the application will hang indefinitely until it is killed and restarted with the failed hardware replaced. This behavior has resulted in the use of checkpointing in order for long-running applications to make forward progress. In this scenario, the results of an application are saved periodically; ideally, checkpoint frequency should be based on the expected mean time to failure (MTTF) of components, given a platform's history and the number of resources involved in an application run. Since the MTTF over a group of resources decreases as the number of resources in the pool increases, the checkpoint frequency also must increase as the number of resources involved in an application run is increased. This in turn limits an application's scalability, since at some point the smallest quantum of work

These authors were supported by the United States Department of Energy, Office of Defense Programs. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

necessary to be accomplished in order to checkpoint cannot be completed before a failure takes place, thus returning the application to the place it began. Additionally, with the increase in heterogeneity introduced by multi-core and mixed CPU architectures and the use of virtualization to provide the illusion of homogeneity, scaling can be further limited by decreases in performance due to overhead and resource contention in an abstracted environment that may appear well balanced.

Intelligent resource utilization is key to enabling efficient HPC applications. The complexity of efficient resource utilization is exacerbated in the commercial cloud paradigm, where knowledge of the underlying resources is deliberately limited (particularly in virtualized environments), the resources are shared among many users, and the physical resources may be heterogeneous.

### *B. Our Approach*

We propose using advanced resource monitoring tools to dynamically characterize the resource and application state, and using the resulting information to optimally assign and manage resources, given the mix of running applications and contention for resources. Such a solution would benefit not only HPC applications, but the full spectrum of applications and usage models for the overall system.

The degree to which resources must be managed, e.g., monitored, analyzed, and controlled, depends on the usage model for those resources. At one end of the spectrum, the resources could be single-CPU servers and the usage as a platform for serial software development. It would then probably be sufficient to provide redundant, mirrored storage resources and a fail-over server. Virtual machines (VMs) created on behalf of users could then be added and deleted until the monitoring of utilization of resources, such as CPU, memory, storage, etc., indicated a resource approaching full utilization. At that point more resources would have to be brought into play or customers would have to be turned away. Another extreme would occur when providing a high performance parallel computing environment for running large, tightly coupled, MPI-based scientific simulation codes: such an environment might consist of hundreds of thousands of multi-core servers with a high speed interconnect such as Infiniband. In this case, a simple standby server is not sufficient, and much more work needs to be put into the monitoring and management system than in the former case.

A common theme across the whole spectrum of use scenarios that the cloud paradigm could address is that the user ultimately should not have to know the details of the underlying hardware and interconnects, but from an abstract point of view should be able to configure a virtual system that presents appropriate physical resources to the application. A facility is needed in order for the user to know if his application is being allocated appropriate resources, either as a post-analysis step to optimize his virtual environment in future runs, or in real time to dynamically optimize the virtual resources allocated to a current long-running application. Such a facility should communicate system attributes to the user, in a way that allows them to make a judgment and modify the configuration as necessary. Additionally, the resource allocation system needs to be aware of the same and possibly additional parameters, in order to make reasonable initial hardware allocation decisions and possibly perform dynamic reconfiguration of resources to adjust to unexpected resource utilization and/or failures.

Figure 1 illustrates an interaction between user, application, monitoring and analysis system, and hardware that can provide dynamic, efficient resource utilization. First, resources are acquired, perhaps upon initial allocation or upon resource or application state change. The application or user can query the monitoring and analysis system for system or application state. The monitoring and analysis system then provides characterizations of the state to be used for resource utilization decisions. In response, the application, user, or resource manager can reconfigure the resources or application to best utilize the resources in light of this state information. An example of alternate utilization strategies is shown in the lower right of the figure. Here each of the blue server blocks consists of multiple cores. In the case on the left, the application needs may be such that it is advantageous for application tasks to be put on different cores, whereas on the right it may be advantageous for tasks to share cores. These scenarios may be as a result of benefits/disadvantages to state information such as shared memory size/bandwidth, internal bus bandwidth, network bandwidth, etc. Each of these hardware resources might be further subdivided into a number of virtual resources, each in turn needing the appropriate level of monitoring and analysis. Hence the system responsible for collecting, analyzing, and reporting this type of information needs to scale to millions of such resources, each having possibly hundreds of associated metrics for evaluation, while being responsive to thousands of users at whatever their expected request rate is.

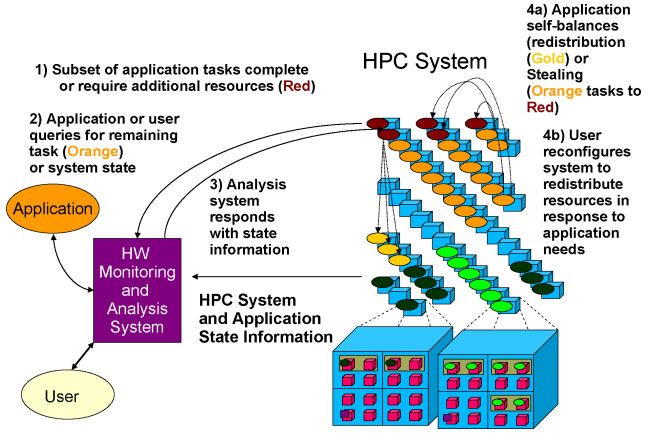


Fig. 1. Illustration of the types of interaction of the user, application, resource monitoring and analysis system, and platform resources in order to dynamically provide and respond to state information. The system seeks to provide efficient resource utilization. The exploded blocks in the lower right illustrate alternative resource utilization in response to differing application needs.

Whatever the mix of scenarios, a monitoring scalability

issue arises because the proliferation of hardware resources is compounded by the number of virtual resources per hardware resource that will be supported. We are currently extending our work in the OVIS project [2]–[6] to encompass not only enterprise computing on traditional HPC clusters but also the cloud paradigm utilized across all scenarios. The OVIS project seeks to provide real-time statistical analysis of large, dynamically evolving data sets. In support of HPC platforms, we continue to investigate the detection of abnormal state behavior and its use as an indicator of potential failure. The reason we believe OVIS is well suited to more general issues in resource monitoring, analysis, and allocation in cloud computing environments is that we have already addressed many of the scalability issues with collection and analysis. We have recently incorporated a multivariate correlation engine into the OVIS tool, described in section III B, designed to assist resource managers with allocation based on a probabilistic assessment of the health of a resource relative to a large aggregation of similar resources.

As an example of the monitoring scalability issue, OVIS collects $\sim 10$ GB of data per day when monitoring just the hardware of Sandia's Whitney cluster of 288 servers with 16 cores each (4600 cores total), and data taken on 10 second intervals. Though not all of the same metric data would need to be replicated on a per-VM basis (for instance, power-supply voltages, component temperatures, and fan speeds), there are many, such as CPU utilization, network counters, memory access and error counters, performance counters, etc., that would; we estimate that roughly 80 percent of the base information applies on a per-VM basis. For this case, if we were running two VMs per core, this would translate into $2.6 \times 10$ GB/day $= 26$ GB/day of information that the monitoring and analysis system would have to process on this modest-sized system. However, this assumes that little to no correlation exists between all of these components. Further analysis may show that due to strong correlations between behaviors of component metrics, many would not need to be collected or analyzed – though scalability would still remain an area of concern.

For cloud computing to mature, we believe it must incorporate evaluation of the underlying resources for application-appropriate use, and expose the evaluation parameters to the user and the application as well. Further, an ongoing evaluation of appropriate resource parameters, both real and virtual, will need to be performed and the results stored for retrospective or real-time analysis and optimization with respect to checkpointing and resource allocation. This type of analysis would also give providers better insight into what knobs they need to expose to users and applications in order to optimize utilization of their resources and hence their revenue.

### C. Outline

In this paper, we first review and discuss related work. We then describe our work in scalable resource characterization and show how it can be used to choose resources appropriate to the expected run-time profile of an application, as well as to provide either run-time feedback to the application or post-run feedback to the user. We present some preliminary findings based on proof-of-concept deployments. Finally, we present planned extensions of our approach to additional aspects of using cloud resources for HPC.

## II. Related work

One of the biggest impediments to scalability of applications in the HPC environment is failure in either hardware or software before an application has time to run to completion; as the application occupies more components, the probability of failure occurring on at least one of these over a fixed time period goes up. The first-order solution has been to implement checkpointing to ensure that forward progress can be made in the face of failures. The main issue with this solution is that writing out the checkpoint data can take significant time and resources and does not scale well. An optimization strategy is then to choose the checkpoint frequency that will maximize net forward progress, given a platform's expected failure rate for the number of resources being used and the expected time it will take to write out checkpoint data given the available storage bandwidth. The goal is to minimize overall run time.

There has been significant work done by a number of researchers to address this problem. Daly has done work in the area of optimizing checkpoint intervals based on the mean time to failure (MTTF) for a system [7]. Additional work has been done by Leangsuksun, Scott and collaborators [8] on making finer-granularity time-to-failure calculations on a per-node basis and using this information to calculate a time-to-failure distribution for an aggregation of such nodes. The goal for both these bodies of work is to minimize the amount of time the application spends writing checkpoints, given that the statistical distribution of failures on a particular platform can be learned and is relatively static. Using these techniques saves time relative to the case of over-aggressive checkpointing and still ensures progress. Work has also been done in the area of predictive analysis by Stearley and Oliner [9] on the Sisyphus project, which seeks to discover correlations of log-file events with both software- and hardware-related failures, and by the OVIS project [3], which looks for correlations of multi-variate hardware state behaviors with failures. The motivation for the latter work was that targeted checkpointing, based upon failure prediction, could dramatically increase the scalability of HPC applications and platforms, since checkpointing all state for the application would no longer be necessary and additionally state would only have to be saved for affected processes when they were deemed destined to fail.

Production-level use of sophisticated resource-state information for robust and dynamic resource utilization is limited. Tools such as Ganglia [10] inspect and maintain state data, but for limited times and on a per-metric basis. Ganglia does not innately provide scalable mechanisms for data fusion of these metrics. Moab [11] supports some level of resource information for use in scheduling, but this is generally unsophisticated state information and not predictive.

## III. Technical Approach

In this section, we describe in brief the OVIS collection and analysis architecture as it applies to the near-real-time monitoring and analysis of large amounts of data, both in and out of band, as well as the initial configuration required to display the physical representation of a cluster in the UI. Next we describe our methodology for characterizing resources described by multiple metrics. Finally we describe how these characterizations can assist in resource allocation decisions.

### A. Scalable Data Collection, Analysis, and Configuration

OVIS works in conjunction with existing low-level data collection mechanisms for capturing information about various aspects of monitored resources in a scalable fashion. For example, data collectors have been written that retrieve information from IPMI, SNMP, LM Sensors, RRD tool, job schedulers, and the proc file system. OVIS uses AVAHI as a discovery mechanism, allowing the data collection entities (sheep) to locate analysis and database entities (shepherds). Through random choice from among a set of advertising shepherds, a sheep will establish a connection with one shepherd, obtain a reference to that shepherd's database node, and insert data directly to that database node. At any point, the shepherd has the ability to reassign the sheep to another shepherd if, for example, the shepherd is over-tasked. If the connection between the sheep and the database node is disrupted for any reason (including failure of the database node), the sheep can connect to an alternate shepherd and to the alternate corresponding database node. Note that getting VM-state-related information from a host operating system would fall to a data collection entity for that host. Thus the data collection scales as the distributed data base grows in membership.

The statistical analyses, including the multi-correlative engine used to perform multivariate probabilistic analysis (described in the following subsection), are designed to scale with database node growth. Scalability is achieved by performing analyses in parallel over all database nodes that contain the corresponding data. Furthermore, analyses used to determine global models, including the multivariate characterization analysis, are robust to failure of shepherd/database nodes, as these analyses are performed within the context of the entire database system (without MPI) consisting of multiple database nodes. In a distributed database system setup, failure of an individual database node will result only in decreased fidelity of the resulting model. Individual comparisons against the global characterizations can be performed only if the data for the particular resource exist; this comparison is not possible with a disrupted connection between the sheep and its corresponding database node. However, once a sheep detects this disrupted connection, the sheep will select another shepherd from the remaining pool. That new shepherd will then perform comparisons on behalf of that resource component.

The configuration for a particular cluster is described by an XML file, including cluster component (e.g., node, rack) specifics such as physical size, relative physical placement, connectivity, labels, functionality, and texture. Details about the metrics being collected for the components are also provided in this XML file. The associations in the XML file define which elements in the pictorial representation can be colored by a given metric.

### B. Resource characterization methodology

In order to characterize resources, OVIS makes use of a multivariate correlative statistics methodology. In this approach, anomalous behaviors are sought by

1) calculating (with "training data") or devising (with "expert knowledge") mean vectors and covariance matrices – and thus, implicitly, a multiple linear regression model – for a set of tuples of variables of interest, and
2) examining how individual observations of these tuples of variables of interest deviate from the aforementioned model; such deviations are characterized in terms of the significance level to which they correspond when the mean vector and covariance matrix are made those of a multivariate Gaussian model. Note that this is directly related to the multivariate Mahalanobis distance computed with the mean vector and covariance matrix.

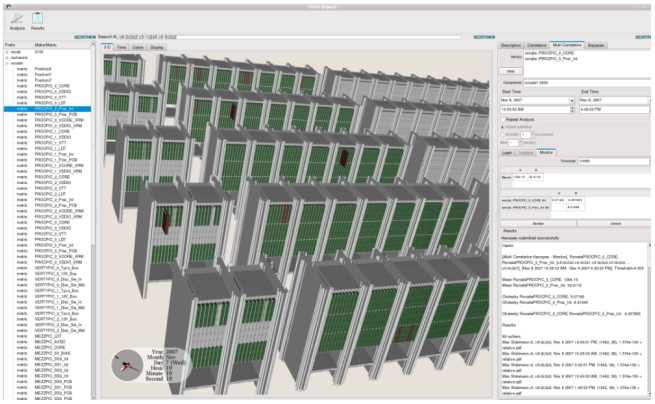

Fig. 2. Actual rendering of the Red Storm platform zoomed in on the partition on which data were taken. The nodes are colored red if below the user-defined probabilistic threshold for being too unreliable and green otherwise. Note that the metrics used in this example are the same two described by Figures 4 and 5. Grey indicates there was no data in the display time window for that resource for the metric being displayed.

For instance, Figure 2 displays a simple use case where only one pair of variables is of interest to the analyst, namely `PROCPIC_0_CORE` and `PROCPIC_0_Proc_Int`, which we will respectively denote $A$ and $B$.

When the first phase of the process described above is meant to be calculated (as opposed to devised, e.g., using expert knowledge), then the "Learn" mode of the statistical engine (called *haruspex*, pl. *haruspices*, in OVIS parlance) is turned on prior to the execution of the haruspex on a set of training data. This is the case in the example of Figure 2: specifically, all observations $(a,b)$ of $(A,B)$ between the specified start and end times (respectively 10:52:02 a.m. and 4:45:02 p.m. on November 8, 2007) on all components called `rsnode`$n$, where $n$ varies between 1 and 3000, are used to "Learn" a model. As a result, a mean vector and a covariance matrix are

calculated, and are available to the user in the "Learn" tab (not selected in the figure).

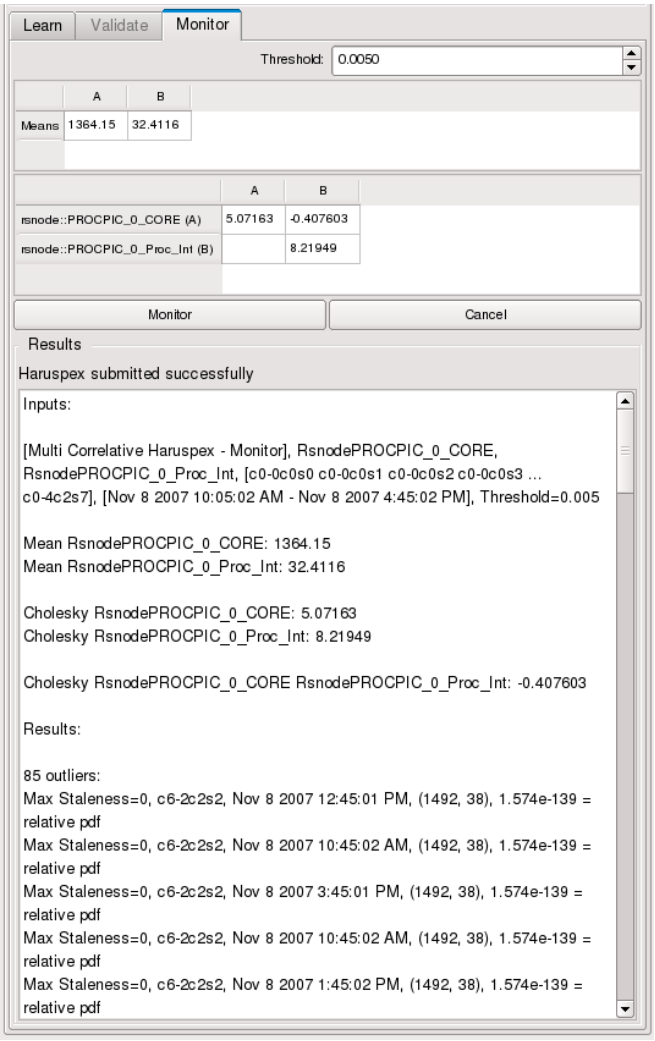


Fig. 3. The interface in Figure 2, zoomed in on the analysis output.

Note that the second phase of the analysis process, called "Monitor", can be performed on either the same data set used to infer a model, or on a different data set. For simplicity, the former option is the case in our running example. Therefore, as illustrated again in Figure 2 and Figure 3, under the "Monitor" tab, one can see the mean vector and Cholesky-decomposed covariance matrix that have been calculated by the haruspex during the "Learn" phase. In particular, the means $\mu_A = 1364.15$ and $\mu_B = 32.4116$ as well as the covariance matrix

$$\Sigma := \mathrm{cov}(A,B) = U^t U,$$

where

$$U = \begin{pmatrix} 5.07163 & -0.407603 \\ 0 & 8.21949 \end{pmatrix},$$

are those of the underlying (bivariate, in this case) linear regression model.

It is beyond the scope of this article to delve into too many details about multiple linear regression models and their relationships to multivariate Gaussian distributions; one only has to know that the underlying linear model is mapped into an $N$-variate (bivariate in our running example) Gaussian model, whose probability density function (PDF) is, by definition,

$$f_X(x) := \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^t \Sigma^{-1}(x-\mu)\right),$$

where $x^t := (x_1, \ldots, x_N)$ is the observation of an $N$-tuple of interest. In our bivariate example, this simplifies into

$$f_{(A,B)}(x) = \frac{1}{2\pi|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^t \Sigma^{-1}(x-\mu)\right),$$

where $x^t := (a,b)$ and thus $(x-\mu)^t = (a-\mu_a, b-\mu_b)$. (Note that the inverse covariance matrix $\Sigma^{-1}$ is computed only once, by means of the Cholesky decomposition.) The argument of the exponential is $-\frac{1}{2}$ times the squared Mahalanobis distance, which is the natural metric associated with the multivariate distribution. The significance level of observation $x$ is defined as the probability (in the Gaussian model) of observing a Mahalanobis distance greater than that of $x$. This significance level is a natural choice of cumulative distribution function (CDF) for the multivariate Gaussian distribution; it ranges from 1 for a central (mean) observation to 0 for observations infinitely far from the mean vector.

With this in mind, we define an outlier as any observation $x$ of $X$ whose significance level is less than a user-specified threshold $\tau$ (typically $\tau \ll 1$). If observations accurately follow the multivariate Gaussian model, then a fraction $\tau$ of observations should meet this criterion. Observations may not follow the inferred model, however, either because the data are non-Gaussian or because the data being monitored have a different distribution from the training data. Nevertheless, the computed significance level is useful in assessing the deviance of an observation.

A simpler description of the significance level is possible in the bivariate case. There, the significance level happens to equal the exponential factor in the Gaussian PDF. This factor can then be described as the relative probability (normalized to the maximum of the PDF), and an outlier can alternatively be defined as an observation $x$ with

$$\frac{f_X(x)}{\max_{\mathbf{R}^2} f_X} < \tau.$$

As shown in the "Monitor" tab of Figure 3, a threshold value of $\tau = 0.005$ was chosen, resulting in 85 outliers being reported by the haruspex, and listed in the lower right text window of the user interface. For example, the first of these outliers corresponds to an observed value of $(1492, 38)$, which, in the context of the underlying model, has a significance level (or relative probability) of $\approx 1.574 \cdot 10^{-139} < \tau = 0.005$, making it an outlier according to our definition. (Such a vanishingly small value indicates that the data are non-Gaussian, since an event with actual probability of order $10^{-139}$ would not realistically occur.) In turn, in the cluster view of Figure 2,

all components evincing outlier behavior at the time shown in the view are colored in red, whereas other components appear in green (data were not collected on the grayed-out components during the time interval of interest).

### *C. Utilizing Statistical Characterizations for Resource Allocation*

Our hypothesis is that the relative health of an underlying hardware component can be described in a probabilistic fashion, based on a behavioral characterization of the component's measurable attributes in comparison with with a sufficiently large aggregation of similar components' characterizations. The motivation for this approach can be seen in Section IV. We seek to utilize such probabilistic characterizations in order to tailor resource allocation decisions to the application requesting them, assuming the user or application can communicate the application's needs. For example, a multi-week 10,000 processor MPI job may require significant checkpoint/restart time, and therefore this job may require the healthiest and hence most reliable resources; however a multi-minute single-processor job being run to test an application bug fix could utilize any available resource, regardless of resource health, and be trivially re-run if the resource happened to fail during the test.

## IV. Preliminary Findings

In this section, we present actual data to motivate our approach in characterizing the relative health of resources (physical servers in this case) using a combination of metrics relating to various attributes of that type of resource. We then show how the OVIS tool presents this information and discuss how it could be used by a resource manager to make more intelligent resource provisioning decisions. Additionally, we briefly discuss the utility of such a visual presentation for system administrators and how, if extended to a web interface, it could give users insight into critical information about their virtual resources as well.

Figure 4 is a histogram of power-supply voltage measurements obtained over a week's period of time on a partition of Sandia's Red Storm platform. This figure is plotted on a log scale to enable viewing of the distributions of these voltages for individual components in comparison with that of the entire set. The histogram of the entire set is shown in green and and is generally distributed in a well-behaved fashion; the magenta measurements are those representative of an actual component that is typical based on the measured mean and variance of the aggregate distribution. We see immediately that the components whose distributions are shown in blue and teal have anomalous behaviors with respect to this aggregate distribution, though in different ways. The blue has a mean roughly 7 standard deviations away from the group mean, whereas the teal has a variance roughly 13 times that of the group. For reference, then, these nodes have been designated in the figure with a shorthand to indicate the type of abnormality – OV (Outlier Voltage) and SV (Standard deviation Voltage), respectively.

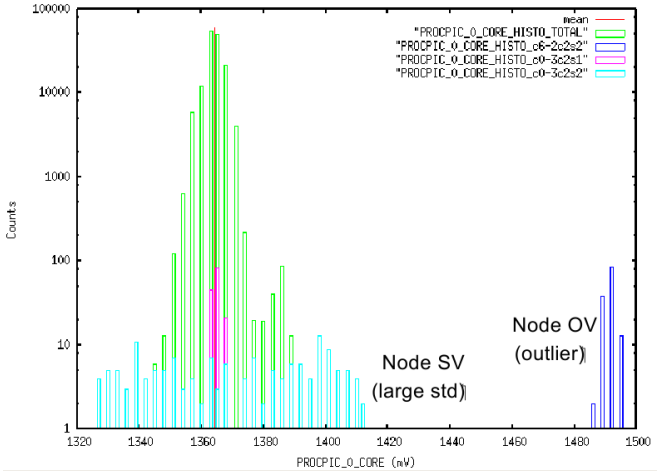


Fig. 4. Histogram (log scale) of a particular power-supply voltage over all nodes in a partition of Red Storm (Cray XT4) for a one-week period. Two nodes whose power supplies exhibit particular abnormal behaviors are called out in the figure.

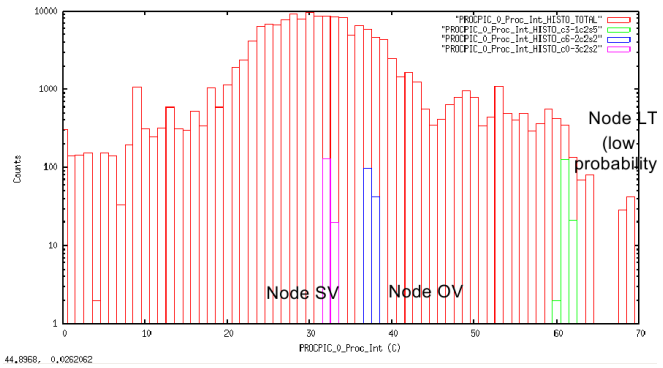


Fig. 5. Histogram (log scale) of a particular component temperature over all nodes in one partition of Red Storm for a one-week period. Nodes exhibiting abnormal behavior in the voltage measurement in the previous figure are relatively well behaved in this temperature metric.

Figure 5 is again a log-scale histogram, but of a processor temperature internal to each server. These data were taken from the same set of components as in Figure 4 and over the same time period. The resources identified in the previous figure as having anomalous voltage values are shown in this figure, with OV now colored in blue and SV now colored in magenta. Note that though these resources were outliers in the voltage metric, they are near the group mean with respect to this temperature metric, with other components lying in the tails of this distribution.

The OVIS multi-correlative analysis engine then "Learns" (as discussed in section III B) a model based upon this data, from which a probabilistic value of any given observation of metrics can be determined. This is illustrated in Figure 6 for a two-metric case using the data plotted in Figures 4 and 5. In this figure, a learned model's surface of relative probability for a two-metric case is overlaid with raw metric observations. The contours of equal relative probability enclose all possible points for which the relative probability is greater than or equal

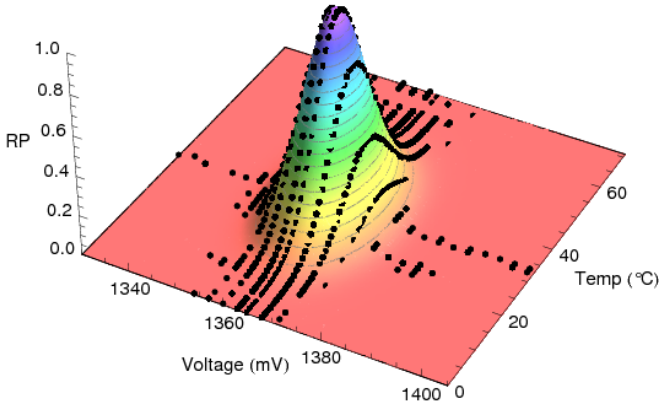


Fig. 6. Three-dimensional plot that illustrates our methodology for assigning relative health to components. Note that these are the same data displayed in the previous histograms, now showing how the values and distributions of two different metrics may be evaluated in this fashion.

to a given value. For instance, in this figure, the points in the outer red zone all lie outside the 5 percent contour. If a system administrator decided that any resource with these two metrics having less than 5 percent relative probability should be considered too unreliable, then any components with metric values lying in this zone could be removed from the resource pool either automatically or manually.

We only use two metrics, in Figure 6 and in our actual example above, to illustrate the basis for our methodology, though it has been extended to arbitrary numbers of metrics as described in Section III.

OVIS's multi-correlative analysis engine, described in Section III, allows the user to choose arbitrary metrics from the metric list for a resource, learn the distribution over a user-specified time interval and for a specified list of resources, define a significance-level threshold, and then (using the learned model and threshold) display probabilistic outliers both visually as a red-colored resource and via text output. Figure 2 is a screen shot of OVIS using its multi-correlative analysis engine to characterize the same set of resources whose metrics for voltage and temperature were displayed in Figures 4 and 5. The blades shown as popped out are those seen to be outliers in each of the histogram plots (blue and teal in voltage and green in temperature). While their data in a single metric do not guarantee that they will be outliers in the multi-correlative case, in this instance the abnormal nodes were sufficiently abnormal even in the single metric to ensure their abnormality in the two-metric correlation. Note, however, that there are other red-colored instances in the figure for which a combination of both metrics led to outliers in the multi-correlative case.

The "Repeat Analysis" mode allows all new data coming in to be compared against a learned model, the significance-level calculation performed, and the results displayed both visually and as text. It would be trivial to share information about the existence of outliers with a resource manager and/or application. Using this same engine, a user could model the behavior of an application in some collectible metrics and look for anomalous behavior that might occur, for instance, if a small subset of resources had other users contending for them. This could in turn drive the user to revise his service-level agreement with the provider, for instance. The kind of web-based interfaces being provided to cloud users for configuration of their virtual resources could present a virtualized representation of the configured resources with colors based on output from such an engine. Whereas the OVIS display uses information about each resource element at every time step in order to color it correctly, a web-based interface might color all virtual elements green except when a threshold crossing triggers a virtual element to update. Even though this would be low-bandwidth to the user, the underlying collection and analysis mechanisms would still have to collect and process the data from the platform and thus must be scalable.

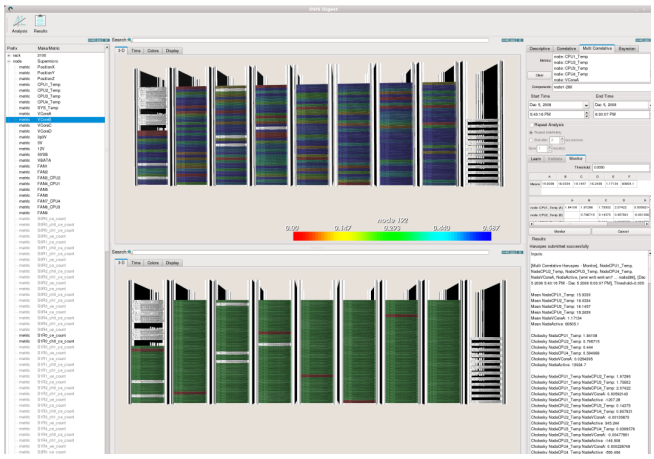

Fig. 7. A rendering of Sandia's Whitney cluster using 6 variables to describe the relative health of a node. The coloring of the node resources in the top representation is by significance level where blue is highest and red is lowest. In the lower pane resources are colored red if below the user defined significance level cutoff threshold.

Figure 7 is OVIS's realistic rendering of Sandia's Whitney cluster, referenced in Section I as an example of the necessity of monitoring scalability. This figure shows the multi-correlative analysis being applied to the cluster for a set of 6 metrics. The top view of the split-screen shows the raw significance level (SL) values associated with each node resource based on the output of the multi-correlative analysis described in section III. The color scheme shows server resources with higher SLs, and hence better health according to our hypothesis, as bluer and those with lower SLs as redder. Using a text output from the tool these resources can be ordered and hence intentionally chosen according to their relative health characteristics for a particular application as discussed in section III C. In this view one can see that there is no real clumping of resources with respect to SL and so we infer no significant environmental effects (at least with respect to the variables chosen). The bottom view shows the output of the monitor mode after a user threshold

has been chosen (in this case 0.005 SL) to demark healthy from unhealthy resources. In this mode OVIS both colors the pictured resources as green to denote healthy and red to denote unhealthy resources and outputs text describing resources lying below the threshold in terms of resource, SL quantities, and time.

## V. Conclusions and Future Work

### A. Conclusions

Cloud computing promises users a computing environment that provides simple, intuitive, just-in-time provisioning of resources (both hardware and software) to meet their application needs. However, there is much complexity that will need to be addressed in the areas of both real and virtual resource management to make cloud computing truly scalable and applicable across the whole range of applications that could potentially benefit from its flexibility and cost savings. The OVIS project addresses the scalable collection and analysis of resource metrics from both component-health and resource utilization perspectives, and hence can contribute to the application-tailored resource allocation of hardware and the subsequent allocation and/or migration of virtual resources on the hardware. Further, metrics describing attributes of virtual resources can be collected and utilized in the same way to provide both providers and consumers of these resources with insight regarding how users/applications are utilizing the resources and hence the ability to optimize their utilization.

### B. Future Work

We plan to build a lightweight web-based application that can tie into the scalable collection and analysis engines offered by OVIS, but can be tailored to show the subset of the environment that the user is interested in, including both physical and virtual devices. A possibility is a generic cluster display (racks with servers) where the number of servers is equal to the number of VMs actually employed; the metrics associated with each VM would represent those from the actual hardware but associated with the user's VMs. Examples of such metrics that could be displayed on the user's virtual cluster include number of VMs currently being hosted by this host, number of VMs that are mine being hosted by this host, percent of resources (memory, CPU, network, etc.) utilized on this host, percent of resources (memory, CPU, network, etc.) utilized on my behalf on this host, etc. A dynamic virtual cluster representation is needed because VMs could be migrated in real time to/from either hardware already in use by a user or hardware not yet utilized by the user. Additionally, virtual resources might be dynamically created or destroyed depending on the nature of the user's request. Finally, we are also planning to investigate other multi-correlative statistical techniques, e.g., multiple non-linear regression.

## References


[1] "Amazon Web Services," http://aws.amazon.com.

[2] "OVIS," http://ovis.ca.sandia.gov.

[3] J. Brandt, A. Gentile, B. Debusschere, J. Mayo, P. Pebay, D. Thompson, and M. Wong, "Using probabilistic characterization to reduce runtime faults on hpc systems," in *Workshop on Resiliency in High-Performance Computing*, Lyon, France, May 2008. [Online]. Available: http://xcr.cenit.latech.edu/resilience2008

[4] ——, "OVIS 2: A robust distributed architecture for scalable RAS," in *Proc. 22nd IEEE International Parallel & Distributed Processing Symposium (4th Workshop on System Management Techniques, Processes, and Services)*, Miami, FL, Apr. 2008. [Online]. Available: http://www.ipdps.org

[5] J. M. Brandt, A. C. Gentile, D. J. Hale, and P. P. Pébay, "OVIS: A tool for intelligent, real-time monitoring of computational clusters," in *Proc. 20th IEEE International Parallel & Distributed Processing Symposium (2nd Workshop on System Management Techniques, Processes, and Services)*, Rhodes, Greece, Apr. 2006. [Online]. Available: http://www.ipdps.org

[6] J. M. Brandt, A. C. Gentile, Y. M. Marzouk, and P. P. Pébay, "Meaningful automated statistical analysis of large computational clusters," in *IEEE Cluster 2005*, Boston, MA, Sept. 2005, Extended Abstract.

[7] J. T. Daly, "Performance challenges for extreme scale computing," http://www.pdsi-scidac.org/publications/, SDI/LCS Seminar Series, Oct. 2007.

[8] Gottumukkala, Liu, Leangsuksun, Nassar, and Scott, "Reliability analysis in hpc clusters," in *Proc. of High Availability and Performance Computing Workshop 2006*, Sante Fe, NM, Oct. 2006.

[9] Stearley and Oliner, "Bad words: Finding faults in spirit's syslogs," in *Workshop on Resiliency in High-Performance Computing*, Lyon, France, May 2008. [Online]. Available: http://xcr.cenit.latech.edu/resilience2008

[10] "Ganglia," http://ganglia.info.

[11] "Moab," http://clusterresources.com.