Deriving Workload Expectations - Monitoring and Analysis Using HPC Job Profiles

Joseph 'Joshi' Fullop IV
HPC-ENV
Los Alamos National Laboratory
Los Alamos, USA
fullop@lanl.gov

Brett L. Layman
HPC-ENV
Los Alamos National Laboratory
Los Alamos, USA
blayman@lanl.gov

***Abstract*—With the growing availability of time series metric data from High Performance Computing (HPC) machines, there is significant potential for using this data to improve the monitoring and analysis of HPC workloads and the systems on which they run. In this paper, we detail the statistical methods for dynamically generating job profiles and workload expectations from this data. This establishes a basis for live job monitoring and enables various methods for detecting aberrant job performance. When a deviation is detected, it can be visualized by displaying the job's metric series plotted on top of its expectation, which is represented as a cloud path. We also demonstrate how to identify system-wide anomalies by monitoring the entire set of jobs on a machine for acute, synchronized deviations. These tools can be applied to a variety of use cases such as shared resource scheduling, benchmark trending, system utilization and planning, and log analytics.**

In the case where jobs are not explicitly grouped by workload type, machine learning techniques can be employed to infer job types based on the shape of the time series data. Jobs are automatically classified, and when a user's workload classification changes, this indicates that something on the system (for benchmark jobs) or in the user's workload has changed.

***Index Terms*—anomaly detection, classification, expectation, HPC, job profile, machine learning, recurrent neural networks, time series analysis, visualization, monitoring**

I. Introduction

A. Disambiguation and Basis

Before we begin, it is important to differentiate between job profiling in terms of embedded libraries that measure how much time is spent in a certain section of the code and job profiling that measures the use of machine resources by the application. We will be discussing the latter. The supercomputers on which this work was done are linux-based and therefore this work may contain terminology specific to that. For example, the term *load* is the common linux OS metric that can be garnered from */proc* or seen in tools such as *top*. Our data consists of metrics collected from compute nodes regarding their health such as: load, memory utilization, bandwidth in, and bandwidth out. There are hundreds of metrics that can be considered, but for simplicity of demonstration we will consider these primary metrics. We used Lightweight Distributed Metric Service (LDMS) from Sandia National Laboratories to gather performance metrics from the compute nodes. The workload organizational data is sourced from the Slurm scheduler. Also in this paper, the term 'cluster' will be used to reference the product of a clustering algorithm (unsupervised learning algorithm), while 'supercomputer' will be used to refer to the machines on which high performance computing is done.

LA-UR-20-27580

B. Job Profiles

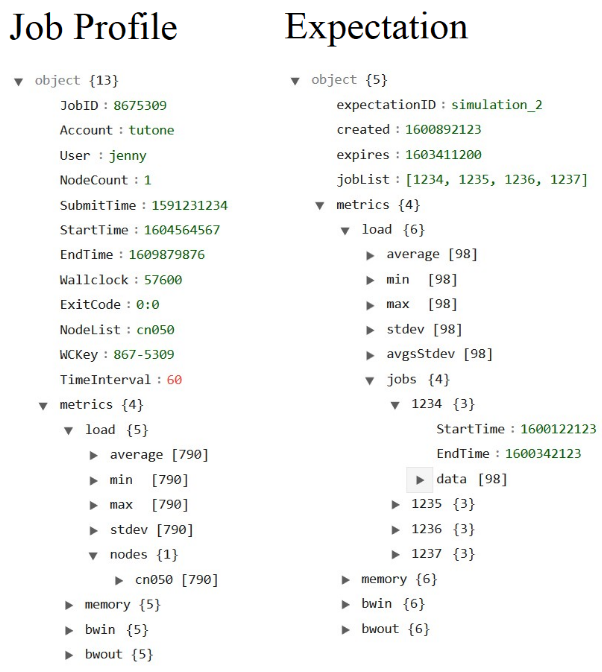
Since time series and organizational data are generally collected and stored separately, joining this data becomes a layered mining process that can put certain types of real-time analysis out of reach for many systems at scale. So to facilitate the type of analytics we had in mind, we created a JSON structure where we could store the entire job profile including basics like job name, start time, end time, node count, etc. as well as time series data (Fig. 1). The metric series are serialized by metric so that rendering them can be done with the least amount of data traversal. Each node's individual metric series is stored under the specific metric branch. Since the data is naturally collected in snapshots, we calculate statistical series per metric for the entirety of the compute nodes as data arrives. These statistics are mean, min, max, count and standard deviation. Using the pre-summarized data enables many things that would otherwise be limited by processing power and bandwidth.

One area where performance may seem trivial is in rendering a graph. Being able to render a job's metric series instantly versus waiting a few seconds for each graph makes a world of difference while exploring the data. Our investigation followed the method of collect, visualize, understand and then automate. That method helped us select the most appropriate techniques and processes in later sections. Instant recall and visualization is also a significant factor when building production-level tools.

II. Job Expectations

A. Motivation

The initial motivation for this investigation was being able to identify and cancel unproductive workloads well before wall-clock termination, and as a result save significant node-hours. The task was to create job profiles, and then determine an expected performance profile (expectation) based upon those profiles. We based these expectations on recent successful runs of the workload in question. This of course will change over time and need to be recalculated with some frequency. We discuss methods for classifying jobs into expectation sets later in the paper. Regardless, significant deviation is a valid selection criteria for identifying jobs that are unlikely to produce reliable results. For example, admins and users are able to determine if a job has died well before the end of its wall-clock time or is behaving in a way that is known to fail. More on identifying known failures in later sections.



Job Profile

```
▼ object {13}
    JobID : 8675309
    Account : tutone
    User : jenny
    NodeCount : 1
    SubmitTime : 1591231234
    StartTime : 1604564567
    EndTime : 1609879876
    Wallclock : 57600
    ExitCode : 0:0
    NodeList : cn050
    WCKey : 867-5309
    TimeInterval : 60
  ▼ metrics {4}
    ▼ load {5}
      ▶ average [790]
      ▶ min   [790]
      ▶ max   [790]
      ▶ stdev [790]
      ▼ nodes {1}
        ▶ cn050 [790]
    ▶ memory {5}
    ▶ bwin {5}
    ▶ bwout {5}
```

Expectation

```
▼ object {5}
    expectationID : simulation_2
    created : 1600892123
    expires : 1603411200
    jobList : [1234, 1235, 1236, 1237]
  ▼ metrics {4}
    ▼ load {6}
      ▶ average [98]
      ▶ min   [98]
      ▶ max   [98]
      ▶ stdev [98]
      ▶ avgsStdev [98]
      ▼ jobs {4}
        ▼ 1234 {3}
            StartTime : 1600122123
            EndTime : 1600342123
          ▶ data [98]
        ▶ 1235 {3}
        ▶ 1236 {3}
        ▶ 1237 {3}
    ▶ memory {6}
    ▶ bwin {6}
    ▶ bwout {6}
```

Fig. 1. Examples of Profiles and Expectation in JSON format.

B. Generating Expectations

Expectations are fairly straight-forward statistical summarizations of past workloads' time series data on a metric-by-metric basis. They consist of a mean, min, max and standard deviation series for each metric. The calculation of expectations is facilitated by our use of arrays for storing time series data in our job profile JSON. The elements of the array are aligned by time, so we can calculate summary statistics on an index by index basis. This process can easily be parrallelized.

Expectation series are calculated using each job's summary of node data (average), and not the individual node series. This allows us to build off of the calculations that have already been performed. For example, we use each job's average node metric value at a point in time to calculate the expectation average. Standard deviation is the only expectation statistic to use more than the node averages in its calculation. We decided to use the standard deviation of all nodes at a particular time to accurately represent the variability in the data at that time, and not just the average of each job's standard deviation, or the standard deviation of the averages. To calculate the total standard deviation across all nodes we used both the average of the nodes and the standard deviation of the nodes across jobs.

Note that we did not need to use individual node metric series for this calculation. For visualization purposes, we decided to also store the average series for each job in our expectation file. This allows the user to readily display all of the jobs that make up an expectation, and makes it possible to check each job for anomalous behavior without having to load all of the individual job profiles. Optionally, for reasons of scale, the individual job series can be left out of the expectation JSON. See (Fig. 1) for examples of the JSON structures.

One special case that is not uncommon is the unique handling of the head node for certain types of applications. These nodes can behave significantly differently than the remainder of the compute nodes. This can also sometimes be seen by the last node and is highly dependent on application architecture.

C. Job Consistency

One notable observation was the consistency of the time series data across multiple runs for certain workloads (Fig. 2). Much of this investigation hinged on there being at least some similarity across runs. It turns out that a significant percent of workloads are highly consistent. Consistency was more common in jobs with smaller node counts, as larger jobs are more susceptible to interference.

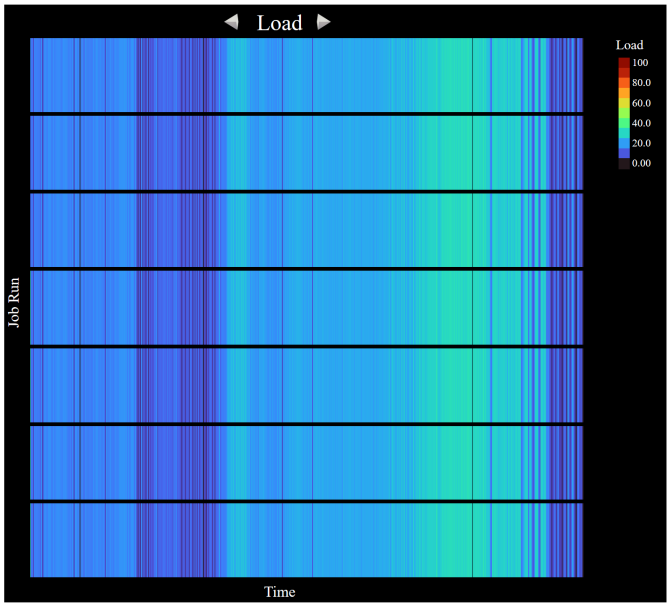


Fig. 2. This plot shows multiple job runs from a particular workload, with each job represented as a horizontal bar. The color of the bar represents the load of the job at different points in time. As you can see, each job run is nearly identical to the next, but not exactly the same. This plot was created with actual single-node jobs.

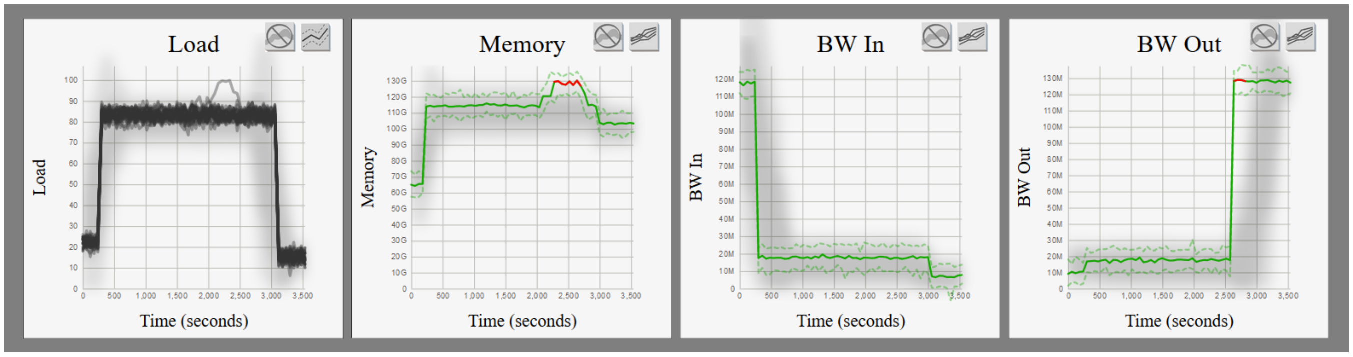


Fig. 3. This figure shows how data from various metrics can be plotted side-by-side to give the user a comprehensive visualization of that job's profile. It also shows the expectation cloud being rendered behind every node's series (far left graph) or a summary of the nodes (remaining three plots). Note that when the average of the summary deviates from the expectation bounds, it can be given a different color to alert the user (second plot from left). These plots were created with mock data for concept demonstration purposes.

D. Web API

After determining what information should constitute a job profile and expectation, we designed a system for managing that data and making it accessible to a user or website. We decided to primarily store job profile and expectation data as JSON files. We chose individual files over a database to avoid lengthy database queries and to simplify data management. For example, if a user wants a particular job, one can simply return them the JSON file. Additionally, using files comes with all of the benefits of data management that the OS provides: permissions, copy, move, pipe, etc. Of course, there are many situations where a database is preferable to files, but encapsulating job data into individual JSON objects enabled us to take advantage of files.

In order to logically maintain our data, we designed a REST API. This API allows the user to create, update, and retrieve job profiles. It also has endpoints for retrieving expectations and anomalies. Using an API makes it easy for the data source to push new job data and for the user to retrieve data. Pushed information is initially stored in a database, and then written to JSON files via double buffering. This allows for multiple processes to write data to files in parallel without bogging down the server. Generating expectations and detecting anomalies also use separate processes. These processes can be triggered by an event, or simply run periodically (i.e. as a cron job).

III. Visualization and Comparison

A. Visualizations

Visualizing time series data with a line graph is a natural thing. We chose to represent the expectation as a cloud path analogous to electron clouds in particle physics. The cloud path is centered around the mean series with the alpha channel decaying as a function of the standard deviation. A Gaussian blur was applied to smooth the appearance of the transitions.

Overlaying an observed job's series on top of it's expectation cloud provides an intuitive way to understand where problems may exist (Fig. 3). Humans can easily identify various anomalies and peculiarities when the data is presented in this fashion.

In addition to visualizing an individual job and it's expectation cloud, a set of related jobs can be displayed as a color map where each job run is represented as a horizontal bar, and the color of the bar represents the value of a metric across time (Fig. 4). This visualization is helpful for identifying trends in job performance, or subsets of a workload type that follow a particular pattern.

IV. Identifying Deviations

A. Aberrant Jobs

There are situations where it simply isn't practical for a human to visually inspect all relevant job profiles and their corresponding expectation clouds. For those situations, we have explored various methods for detecting when deviation from the expectation reaches an anomalous level.

One technique is to measure the average deviation across the entire course of the job. This technique is likely to pick up on major anomalies, such as when the load drops to zero for a quarter of the job time, but smaller point anomalies will likely be missed. On the other end of the spectrum, one could simply look at the deviation for the latest point to detect recent anomalies. This technique is likely to have many false positives, due to the high number of comparisons, but it could be used for immediate notifications about live jobs. Weighting data by recency can be a nice compromise between these two methods.

A technique that we found to be particularly useful is maximum window deviation. This measurement is taken by choosing a specific interval length, let's say 20 minutes, and then checking each 20 minute segment of the job to see where deviation is the highest, and saving the highest deviation as our maximum window deviation for that job. We have anecdotally observed that requiring a job to fail multiple maximum window deviation tests with different window sizes has been a fairly reliable way of detecting outliers.

Once deviation has been measured for a collection of jobs, we can statistically compare each job to the distribution of other jobs in that set. In our case, we chose to use a Z-score and its corresponding p-value, although there may be other

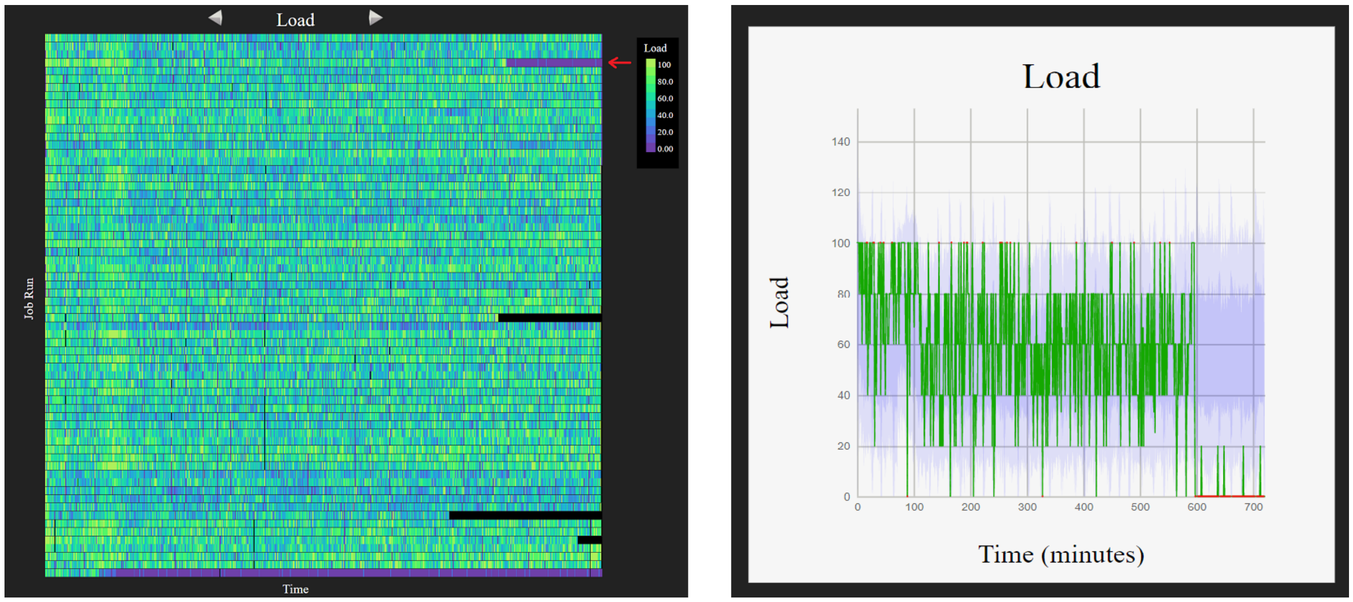


Fig. 4. The plot on the left shows all of the jobs of a particular workload type, with each job represented as a horizontal bar. The color of the bar represents the load of the job at different points in time. As you can see, five of the jobs either failed or dropped to a load of zero. The plot on the right shows the load curve plotted over the expectation cloud for one of those failed jobs (marked with the red arrow). These plots use actual load data taken from jobs run on a supercomputer.

statistics that work better for particular sample sizes. When measuring deviation for multivariate data (multiple metrics), we used Fisher's method to combine p-values. This creates a new p-value, which can be smaller than the p-values of each metric, showing a kind of "sum is greater than its parts" significance. Formal experiments will be needed to determine which statistical techniques are best suited for a particular purpose. Here we are simply presenting some basic techniques that have provided us with some promising preliminary results.

One issue that we have encountered is that of false positives. Accordingly, we have had to adjust our alpha threshold value to be fairly low (e.g. p ¡ .0001). This could be partially due to the nature of time series data. Often there are many data points to compare in the series, and if enough comparisons are run, then some of those comparisons are bound to be significant. In addition to lowering our alpha value, we have also mitigated these false positives by requiring that jobs reach significance on multiple deviation metrics before being marked as anomalous.

B. Single Node Anomalies

At this point it should be apparent that there are a number of ways to determine when a job is currently out of expectation. Consider the case where a job matches its expectation profile with a minor exception period. This anomaly has a high likelihood of being due to some event external to the application. This means that job profiles and expectations can be used to aid in the monitoring of the supercomputer. Additionally, the anomaly can be pinpointed to the exact time and node on which something occurred that impacted the job. This can aid heavily in guiding root cause analysis when examining system logs.

C. System-Wide Events

In the course of monitoring live jobs for deviations, we traverse all of the currently running jobs and compare them to their expectations. Those found to fail the aforementioned deviance tests are flagged for human examination, or when confidence is sufficiently built, acted upon automatically. When a large percent of jobs deviate at the same time, this can be indicative of a significant system-wide event.

We made a compelling observation while examining sets of runs for a particular workload. We noticed that job runs from this workload were extremely consistent in their load metric, except that they all had a similar-looking dip at some point in their series. Some of the runs had this dip at one point in the course of the job, while others had the dip at another point, or yet another point. This brought up the question of why the dip was aligned across some of the jobs and not others. Initially this was curious, but then we realized that these jobs did not all start at the same time. When the graphs were time shifted based on their start times, all of the load dips aligned perfectly (Fig. 5). This lead us to find that an external event may be larger than something impacting a single node.

Picking up on system-wide events can be incredibly important for helping users with workload issues. If these events aren't identified, then a user might waste a significant amount of time trying to fix the application, when in reality the issue was caused by some system event.

In addition to visualizing a time-shifted set of jobs from one user's workload, we've also developed tools for displaying jobs from multiple users (Fig. 6), and even visualizations of every job on a system for a particular time interval (Fig. 7). These visualizations not only make system wide events more apparent, but also provide an intuitive holistic view of the system. Additionally, pan and zoom controls make it easy to focus in on particular events or scan across time. If job expectations are available, it's also possible to display deviation rather than raw metric data in this visualization, and look for bands of high deviation (Fig. 7).

When exploring this data, we found a large amount missing data. This brings up another important use for these tools: assessing the reliability of data obtained from monitoring systems. If large segments of this data are missing, then other tools that rely on the data could be ineffective for that time.

With these system-wide observations in mind, we devised a more direct computational method of identifying impactful system-wide events. If we accumulate the number of standard deviations that a metric is currently out of expectation and then normalize it to the number of nodes currently running jobs, we can use a threshold to discern system-wide events.

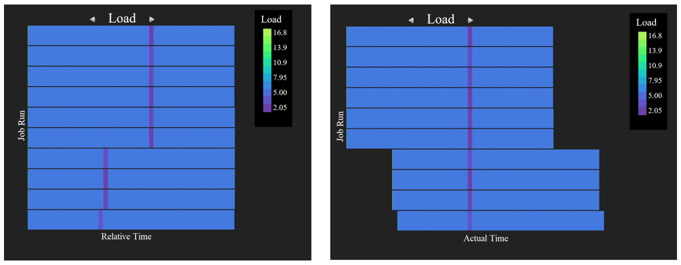


Fig. 5. These plots show multiple job runs with each run represented as a horizontal bar, and where the color of the bar indicates load values for the job across time. The graph on the left shows the bars aligned relative to when each job started, whereas the graph on the right shifts jobs according to their actual start time. These plots were generated with actual data from single-node jobs.

V. Job Classification

A. The Need to Classify

Up to this point, we have assumed that jobs are explicitly grouped into different workload types, allowing for expectations to be generated from jobs of that type. However, it isn't always the case that jobs will be labeled in this way, or they may be labeled incorrectly. For example, a user might keep the same name for a job after making fundamental changes to how it runs, and subsequent runs of the job will be marked as anomalous relative to the expectation. In this section of the paper, we will explore a machine learning technique for inferring job type based on a job's time series data.

Machine learning techniques have several advantages over other techniques in addressing this problem. A simple approach to this problem would be to match or cluster (if numerical) top-level job attributes. While this technique has utility in narrowing the scope of which jobs need to be differentiated from each other (i.e. jobs from the same user with the same node count and requested machine time), it isn't ideal for making finer distinctions. Another approach would be similar to what we outlined above for anomaly detection: using average deviation to group jobs. With average deviation as a distance metric, a clustering algorithm could be used to place jobs into groups. Although this technique uses more detailed data about a job, it still has major pitfalls. For example, if the time series is offset by even a small amount, it could result in a significant increase in deviation, despite the sequences being nearly the same. A similar situation occurs with stretching or compressing the series by a small amount. In other words, the algorithm isn't comparing based on sequence, but rather based on individual time points. There are ways to remedy these issues, but an algorithm that is able to encode sequential information avoids the issues altogether.

B. Machine Learning Approach

Recurrent neural networks (RNNs) are capable of encoding sequential information by maintaining a state that retains past information and combines it with new information. In fact, research has shown that RNNs can be successful at a variety of sequential modeling tasks, and have key advantages over other sequential models such as Markov models [2]. More specific to our purposes, RNNs have been used to effectively model time series data [3]. In this respect, RNNs are a promising candidate for encoding job metric data and using it to distinguish workloads from one another. Encoding jobs based on how their metrics change over time helps to avoid grouping jobs that appear similar but are fundamentally different on a functional level. Given training data with stretched or offset sequences, a RNN is capable of recognizing that all of those sequences fall under the same category of job, represented by nearly identical encodings, except for a few numbers to adjust the offset or scale.

The specific type of RNN that we chose to test is a Long Short Term Memory Auto-encoder (LSTM-AE) network. LSTM networks are known to have several advantages over simple RNNs, particularly in how they address the vanishing/exploding gradient problem [4]. It was necessary to use an auto-encoder because of the absence of labeled data (unsupervised learning task). This algorithm works by converting variable-length time series data into a fixed-length vector (via the first recurrent layers), then using that vector to recreate the original input (via the last recurrent layers). In other words, it learns a fixed-length encoding for the variable-length input data. For example, it could learn to produce the same output given a job that has completed or is only half way through, because it has learned to infer what the full output should look like given the initial sequence. After learning the encodings for various jobs, those encodings can be used for data analytics techniques such as: clustering, visualization, Principal Component Analysis (PCA), and more.

We propose using learned clusters of encodings to classify jobs into expectation groups. One advantage of compressing job data into encodings is that the encodings only retain the most essential information necessary to approximate that job's time series. For example, if there are essentially two job types, then the encoding only needs to indicate which type the job belongs to in order to produce a reasonably accurate time series. Technically, only 3 bits would be required to distinguish

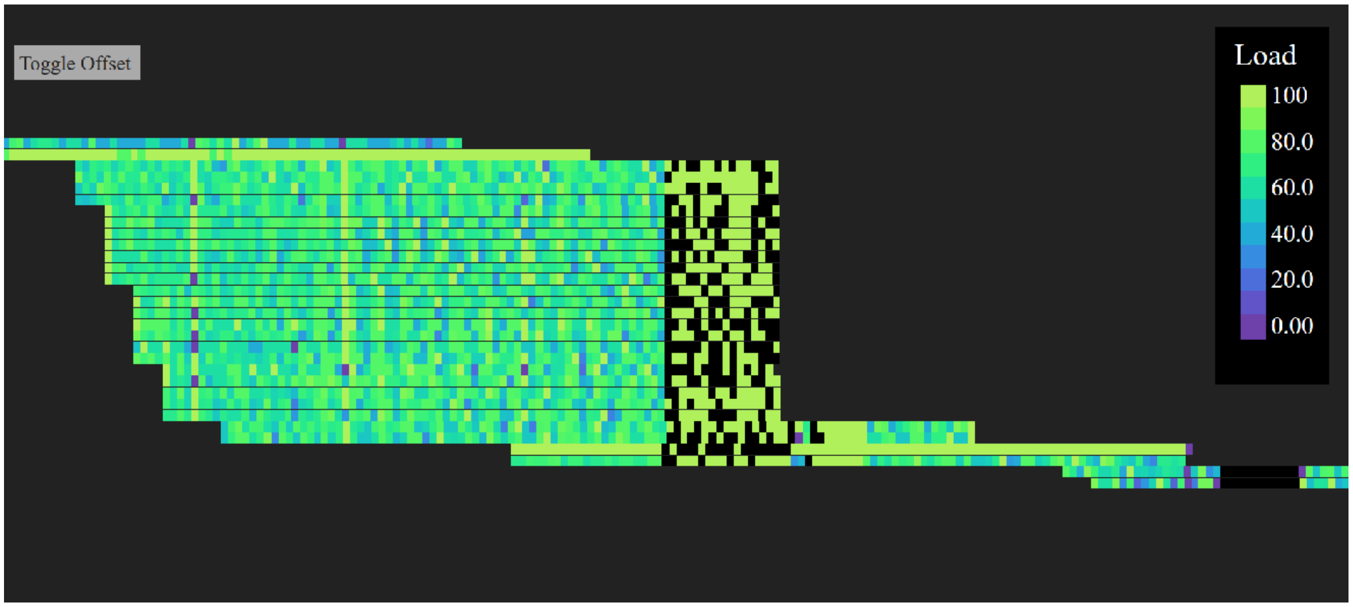


Fig. 6. This color map shows jobs, which are represented as horizontal bars. Each bar is shifted horizontally according to the job's start time. The load values for a job are mapped to unique colors and displayed across time. Note how there are a number of jobs which start at different times, but all seem to end at the same time. The load for the jobs shows some modest variability up until the end, where there is a period of high load mixed with missing data (shown in black). The simultaneous occurrence of this segment of seemingly disruptive activity makes one wonder if there was a system-wide event that caused most of these jobs to terminate prematurely.

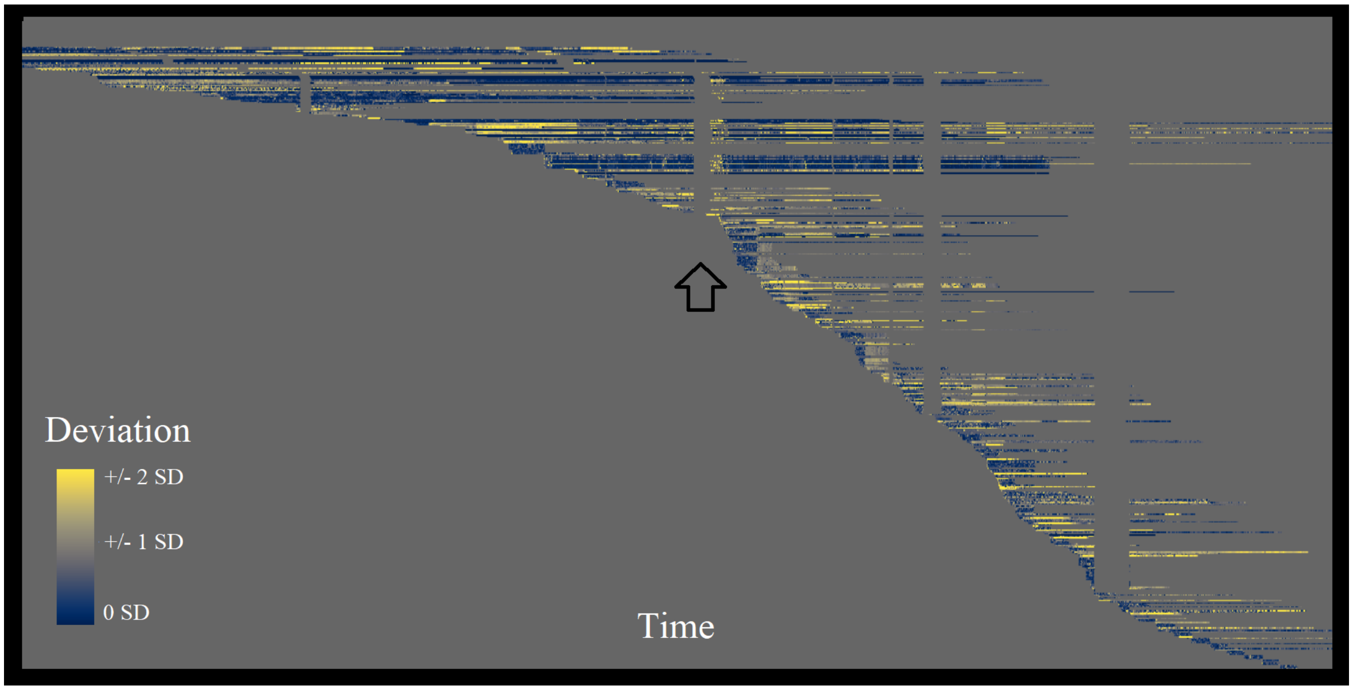


Fig. 7. This plot shows all of the jobs run on a system over the course of a day. Each job is represented as a horizontal bar, where the bar is shifted according to the job's start time. The color of the bar represents the deviation of that job from it's expectation in standard deviations. The sign of the standard deviations is not used in this mapping, just the distance from the mean (absolute value). Values are capped at 2 standard deviations. Missing data is displayed as completely transparent. Note the vertical bands of high deviation (yellow) across multiple jobs. These bands suggest a system wide event in that they effect most of the jobs on the system simultaneously. The black arrow points to one of these events that involved missing data followed by low load (which is now displayed as high deviation).

between 8 job types, although using the smallest possible encodings may not be the most efficient way to distinguish job types in practice.

C. Investigative Process

To demonstrate the utility of the LSTM-AE for job classification, we set up an exploratory experiment where we trained the LSTM-AE on three known but unlabeled job types. We then clustered the encodings for the training and test jobs to see if three distinct clusters emerged, and if the test jobs fell into the correct clusters. We trained on 5-7 jobs of each type, and tested on 2-5 jobs from each type. Jobs of the same type had similar but not identical time series. We used a 13 layer LSTM auto-encoder to learn job encodings. The network was trained for 2,000 iterations with a batch size of 10,000. Note that each training example in the batch was derived from the original 5-7 training examples by randomly selecting an example, then cropping the time-series to be between one quarter and the full length of the job (as determined by uniform random number). This teaches the network to recognize jobs that haven't finished yet. The ADAM optimizer was used to adjust network weights. We chose the ADAM optimizer because it often converges faster than other commonly used optimizers [1]. Test jobs were not used in the training process. After training, we used PCA to reduce the dimensionality of the encodings to 2 dimensions, accounting for approximately 99.98% of the variability. We then used DBScan to group the job encodings into clusters. We found that the jobs clustered neatly into 3 distinct clusters, where all training and test data was clustered correctly according to its job type (Fig. 8). We also found that feeding cluster centers into the decoder portion of the network resulted in time series that resembled a job in that cluster (Fig. 9). Although these initial results are encouraging, they do not represent a formal experiment, and further investigation will be necessary to explore the utility of this algorithm in various contexts and compared to other algorithms.

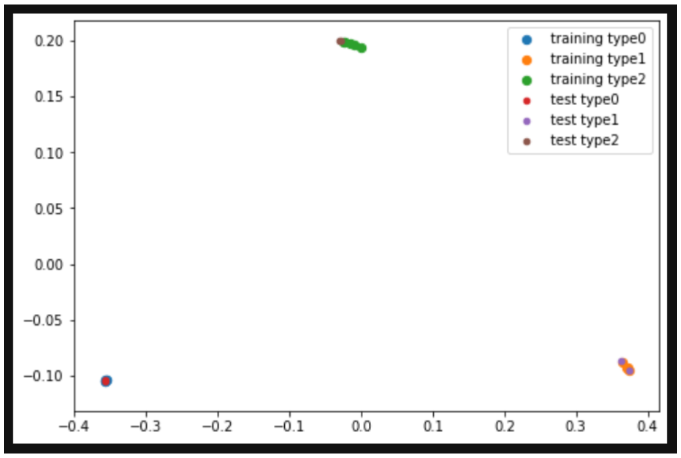


Fig. 8. After reducing the dimensionality of job encodings to two dimensions, we plotted the encodings for both training and test data. As you can see, three distinct clusters emerged: one for each job type. All test jobs fell into the correct clusters.

One might make the connection between an expectation and a series generated from a cluster center. These are similar to the extent that we can use the expectation to validate the generated series. However, we cannot use the generated series in the same way we use the expectation because the generated series does not contain standard deviation information.

D. Limitations

Like all inference techniques, the use of an LSTM-AE to classify jobs is not without its limitations and draw-backs. Our neural network and deep recurrent networks in general can be computationally expensive and require long periods of training to learn new sequences. Training these networks on a single CPU can be impractical. However, parallelizing the algorithm to run on a GPU (straightforward with Tensorflow) or on a supercomputer, can speed up the algorithm by several orders of magnitude. The general process of classifying jobs could also be sped up by first partitioning jobs based on top-level attributes and then running a LSTM-AE on those subsets to make finer distinctions.

Another notable limitation relates to how the algorithm handles series that it has not seen before in the training process. The algorithm will often make a "best guess" about which encoding cluster the job belongs to, rather than encoding the job as an anomaly, or placing it in a new cluster. In some situations, this could be helpful, such as when you want to place a job that failed in the same cluster as jobs of that type that succeeded. However, this could be misleading if you don't implement an additional anomaly detection step (such as the deviation-based technique described above) after classification. This issue is resolved by simply training the algorithm on the new job, so that it learns an appropriate encoding for that job. Given enough training examples, the algorithm will learn to classify specific types of job failures in addition to job types.

One final limitation relates to how the algorithm learns when within-type differences outweigh between-type differences. The encodings learned during the training process represent the most important information needed to recreate a job's series, and if there is high variability within a type, a significant portion of the encoding may be devoted to distinguishing within-type differences, which could in turn lead to poor between-type clustering. That said, what constitutes within-type vs between-type could be an arbitrary distinction when no predetermined types exist. If a user's jobs start to fall out of the original cluster, that could represent a meaningful change to either the system or the job itself. In fact, one area in which this algorithm could be particularly effective is with benchmark jobs. These jobs should be fairly consistent and cluster nicely. When an anomalous run occurs and is adequately trained on, a new cluster will emerge indicating that something on the system has changed.

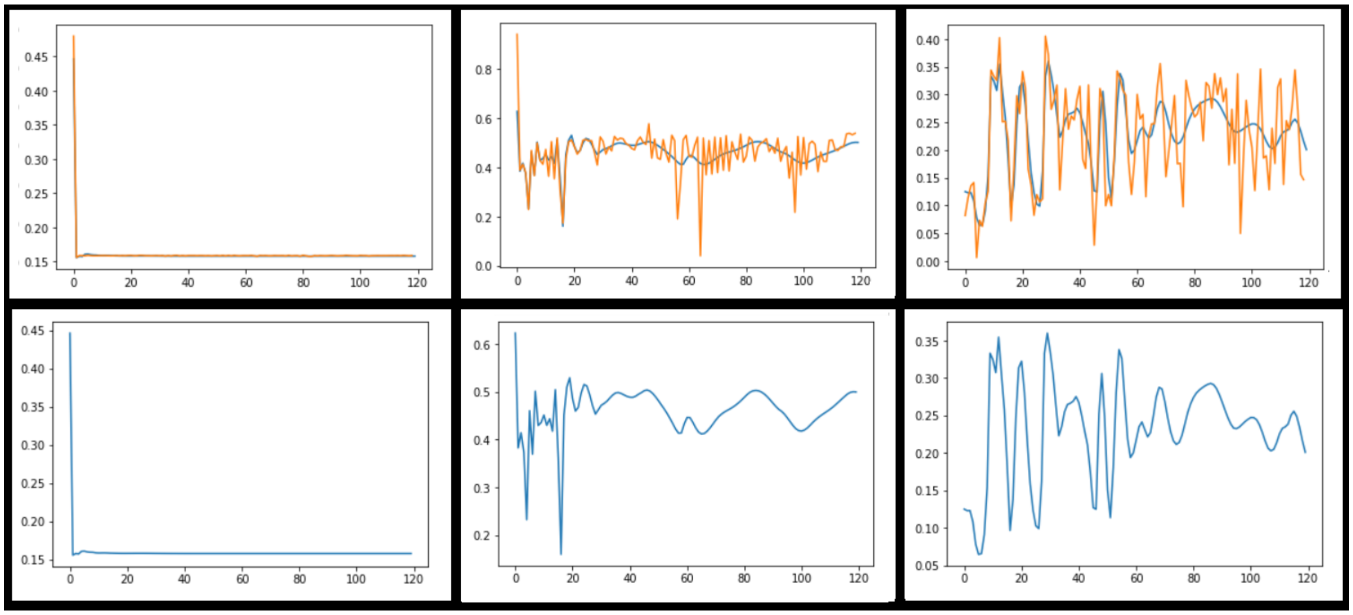


Fig. 9. The top three images depict the generated series (blue) overlaid with the actual time series that they were generated from. The bottom three images show series that were generated from the center (average) of each encoding cluster and were not generated from a specific job.

VI. Other Uses and Areas of Impact

A. System Benchmarking

Job profiles can provide more utility that just depicting how an application utilizes the resources allocated to it. When a benchmark application is run with regularity, an administrator can see when something had changed on the supercomputer that has affected the performance of the workload. This is particularly useful after system or programming environments updates. This can also be useful for measuring long term performance degradation (Fig. 10).

B. Resource Scheduling

Expectations have a number of uses outside the scope of deviance detection. A scheduler can consider a job's expected resource demand when determining if it has the budget available to add that particular job starting at that time. For example, a job may not be allowed to start if its power draw would cause the budgeted peak to be exceeded at any point in its life when added to the other workload's expected draw. It can also be used to regulate shared resource loads like parallel file system writes in the same way.

C. System Utilization and Planning

It should be obvious that profiles can be used to verify that applications are using available resources to the fullest extent. Users can identify a potential limiting factor and tune their application accordingly. From a managerial perspective, profiles can be used in aggregate to determine how a supercomputer is being utilized. Then based on some weighted analysis, determine what attributes (memory, CPU speed, etc.) would better serve the user community when planning the next supercomputer. It could also be used in the acceptance process and to compare an application's performance across supercomputers.

D. Log Analytics

One of the most difficult problems in the area of log analytics is the validation of detected anomalies. Even after an anomaly is discovered in the logs, there needs to be a determination of how it impacts the workload. So frequently identified anomalies are meaningless. Log analysts often request a list of faults to use in this endeavour [5], [6]. But oftentimes, a list of failed jobs is the only data available, and this is usually not sufficient or precise enough for their purposes. Using our proposed system-wide event detection technique, we can add the impactful events back into the log stream and simplify the validation process. Or we could use it further to drive a log analytic system as described in *A Diagnostic Utility For Analyzing Periods Of Degraded Job Performance.* [8] that would surface anomalous log message types during the periods of mass workload deviation.

VII. Summary

Cray has invested significant time and effort into creating an infrastructure to provide customers with monitoring data. This has come at the community's request, which also includes questions like *What is normal?* and *What thresholds should we be alerting on?* With the complexity of such systems those are very difficult questions and require ample system experience to even begin to address. With this expertise requiring time to acquire, answers to many of these types of questions will not be available upon the delivery of the Shasta era machines. The same would hold true for any future system employing novel hardware, software or architecture. With domain expertise also

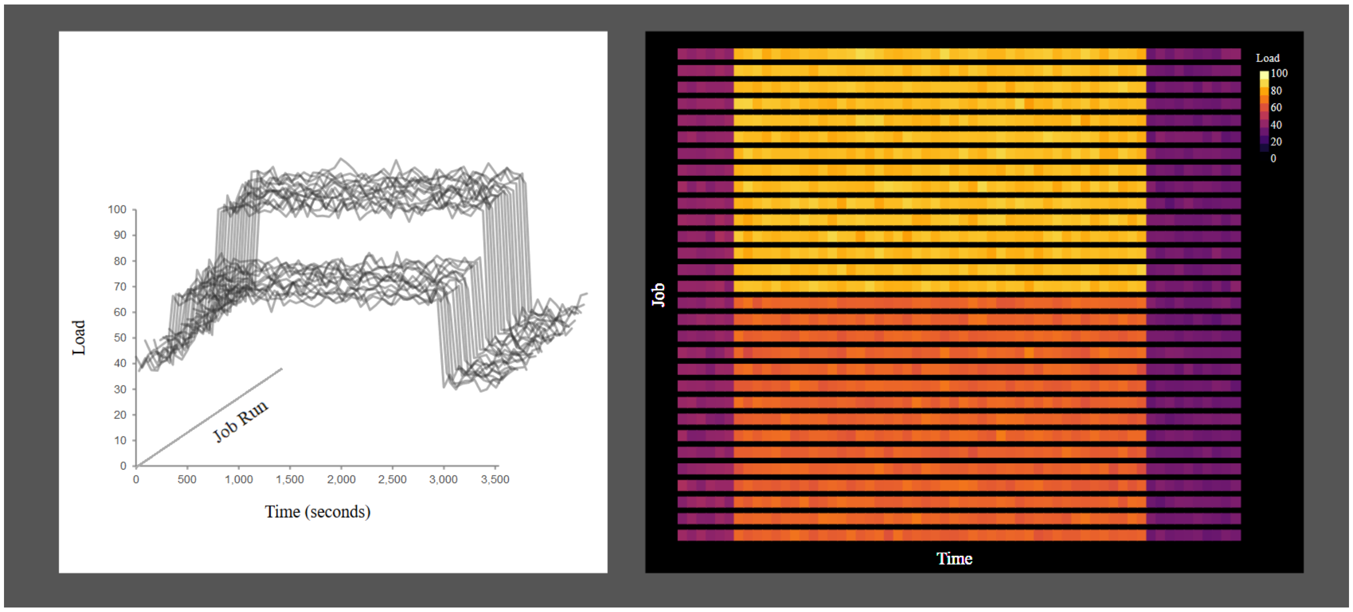


Fig. 10. Using mock data, these plots show multiple benchmark jobs overlaid with an offset (left) and as bars in a color-map (right). You can see that there was a substantial shift in the load values for a portion of the job, indicating that something may have changed on the system.

being difficult to convey, maybe we should forego reliance on it in favor of more system agnostic tools.

Ultimately, the importance or significance of events can be gauged by how they impact the workload. Being able to identify and measure this impact and then quickly associate relevant log messages greatly reduces the reliance on domain expertise to understand and expeditiously address problems. This method can be considered an early version of automated root cause analysis.

Over the course of this paper, we have discussed a set of data management and analytics techniques to facilitate the study of workloads on supercomputers. We started with only job records and time series data from compute nodes. From there we created a job profile, and then an expectation using only those profiles. We then used those expectations to monitor live jobs and implemented various methods for determining deviations. Machine learning was employed to classify workload automatically in lieu of relying on user-based tagging. We further considered the cumulative deviation across all workloads to identify system-wide events.

Using these mechanisms, we are able to better identify and understand when things change on a large-scale system. Triaging application issues into application error or system error is a fundamental part of running and maintaining supercomputers. Early identification of failed or expected-to-fail jobs provides us with valuable, actionable information that allows us to notably improve our workload throughput. Also, identifying if a job was impacted by a system-wide event can save our staff countless hours of investigation.

There are many future areas to explore and many tools to be built from the basis we have established here. We look forward to discovering new techniques and creating applications using these job profiles and expectations.

REFERENCES

[1] D. P. Kingma, J. L. Ba, "ADAM: a method for stochastic optimization," ICLR Conference, 2015.

[2] Z. C. Lipton, J. Berkowitz, "A critical review of recurrent neural networks for sequence learning," arXiv preprint arXiv:1506.00019, 2015a.

[3] G. Petneházi, "Recurrent neural networks for time series forecasting," arXiv preprint arXiv:1901.00069, 2019.

[4] A. Sherstinsky, "Fundimentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," in Physica D: Nonlinear Phenomena, vol. 404, March 2020: Special Issue on Machine Learning and Dynamical Systems.

[5] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer. "Lessons learned from the analysis of system failures at petascale: The case of Blue Waters." In Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on, pages 610–621. IEEE, 2014.

[6] A. Gainaru, F. Cappello, M. Snir, and W. Kramer. "Fault prediction under the microscope: A closer look into hpc systems." In SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pages 1–11, Nov 2012.

[7] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. "Failure prediction for hpc systems and applications: Current situation and open issues." Int. J. High Perform. Comput. Appl., 27(3):273–282, August 2013.

[8] Joshi Fullop and Robert Sisneros. "A Diagnostic Utility For Analyzing Periods Of Degraded Job Performance." Cray Users Group, 2014.