# Chapter 1
# Performance Analysis Tool for HPC and Big Data Applications on Scientific Clusters

Wucherl Yoo[1], Michelle Koo[2], Yi Cao[3], Alex Sim[1], Peter Nugent[1,2], Kesheng Wu[1]

**Abstract** Big data is prevalent in HPC computing. Many HPC projects rely on complex workflows to analyze terabytes or petabytes of data. These workflows often require running over thousands of CPU cores and performing simultaneous data accesses, data movements, and computation. It is challenging to analyze the performance involving terabytes or petabytes of workflow data or measurement data of the executions, from complex workflows over a large number of nodes and multiple parallel task executions. To help identify performance bottlenecks or debug the performance issues in large-scale scientific applications and scientific clusters, we have developed a performance analysis framework, using state-of-the-art open-source big data processing tools. Our tool can ingest system logs and application performance measurements to extract key performance features, and apply the most sophisticated statistical tools and data mining methods on the performance data. It utilizes an efficient data processing engine to allow users to interactively analyze a large amount of different types of logs and measurements. To illustrate the functionality of the big data analysis framework, we conduct case studies on the workflows from an astronomy project known as the Palomar Transient Factory (PTF) and the job logs from the genome analysis scientific cluster. Our study processed many terabytes of system logs and application performance measurements collected on the HPC systems at NERSC. The implementation of our tool is generic enough to be used for analyzing the performance of other HPC systems and Big Data workows.

## 1.1 Introduction

Large science projects have been relying on thousands of CPUs to compute terabytes or petabytes of data [18][32]. This chapter studies the challenges of analysis on

[1]Lawrence Berkeley National Laboratory, Berkeley, CA, USA, [2]University of California at Berkeley, Berkeley, CA, USA, and [3]California Institute of Technology, Pasadena, CA, USA

large amount of monitored performance measurement data from the cluster system, and tackles the challenges by providing a performance analysis tool. Many HPC applications are built to generate and analyze terabytes or petabytes of data, and they often require running over thousands of CPU cores and large amount of data accesses, data movements, and computations. HPC applications running on HPC platforms include parallel applications or high throughput computing applications, and these applications could involve Big Data workflows. The job executions from the complex workflows generate a large volume of measurement data over time. Due to the complexities of the job executions on the large number of machines and large amount of data, it is challenging to identify bottlenecks or to debug the performance issues in HPC applications and scientific clusters. Understanding the performance characteristics of the complex scientific workflows managing thousands of concurrent operations and debugging their performance issues are challenging for various reasons. The concurrent data accesses may compete for shared data storage and networking resources with each other on the system. The performance characteristics on the current generation of the storage hardware and memory hierarchies are sometimes unexpected due to the complexities. Unexpected delays can be introduced by the temperature-based throttling mechanisms on the modern CPUs, which reduce the clock rate to decrease heat production. It is common for large parallel jobs to experience mysterious performance fluctuations. To address these challenges and to help understand these performance fluctuations and diagnose performance bottlenecks, we have developed PATHA (Performance Analysis Tool for HPC Applications) [40] for HPC applications and scientific clusters using a state-of-art big data processing tools .

Our tool can ingest system logs and application performance measurements to extract key performance measures, and apply the most sophisticated statistical tools and data mining methods on the performance data. It utilizes an efficient data processing engine to allow users to interactively analyze large amounts of different types of logs and measurements. Using PATHA, an interactive exploration of the performance measurement data is enabled for the user's understanding about the performance of their own applications. A big data processing framework, Apache Spark$^{\text{TM}}$ [43] is employed in the backend to distribute and parallelize computational workloads for analyzing large amounts of performance data. Spark$^{\text{TM}}$ can utilize in-memory processing to reduce an overhead of loading data from disk. Compared with other big processing frameworks such as Hadoop, Spark$^{\text{TM}}$ fits better for PATHA to conduct performance analysis combined with in-memory computations by reducing loads and stores of intermediate results on disk. PATHA can identify performance bottlenecks through outlier detection and other data mining techniques through the extensive analysis capability of Spark$^{\text{TM}}$. PATHA further provides interactive visualization of these bottlenecks and their dependencies, and allows quick integration of the new performance information as it gathers from the newly generated log files.

For case studies, we have worked with the Palomar Transient Factory (PTF) [23][29] application and job logs collected from Genepool cluster [1] for genome analysis. We have used PATHA to analyze application performance of the PTF application with the measurements collected on the NERSC Edison cluster. We have also analyzed system performance to identify job performance outliers from the logs of Genepool cluster. We believe that PATHA is applicable to other analysis cases for conducting performance analysis and bottleneck detection, and these example case studies are representative use cases. It is generally applicable to the combined multiple data sources such as application logs and cluster logs from schedulers, sub systems of clusters, or monitoring tools.

The PTF application is a wide-field automated survey that records images of variable and transient objects in the sky [23][29]. Images from these cameras are sent and stored to the NERSC Edison cluster for processing through the near real-time image subtraction data analysis pipeline. In each processing step, the timestamps of the execution were recorded in the database. As the PTF analysis processing pipeline has been optimized, its performance analysis to find hidden performance bottlenecks is particularly challenging. In addition, queries on the database need to be minimized for severe overhead on the production database shared by many users. Through our study with PATHA, we were able to identify and to optimize hidden performance bottlenecks and inefficient operational steps, without incurring large database overhead.

The Genepool scientific cluster produces large job logs from a large number of nodes and multiple parallel task executions, and it is challenging to analyze and extract meaningful information from the job logs due to the complexities. Many performance-related fields in the logs are correlated to each other, and jobs interact in the task executions. Using PATHA, we were able to analyze system performance in an efficient and user-friendly way, to extract interesting information about system performance, and to identify performance outliers. We believe that PATHA can analyze the performance of other scientific workflows as well as cluster systems using the application logs and cluster systerm logs.

The contributions are:

- the design of bottleneck detection methods in PATHA, e.g., execution time analysis and data dependency performance analysis
- the development of PATHA to handle different types of measurements from scientific applications and HPC cluster systems
- the evaluation of PATHA using a big data application such as PTF and large-size job logs of a scientific cluster

The rest of the chapter is organized as follows. Sec. 1.2 presents related work. Sec. 1.3 demonstrates the design and implementation of PATHA. Sec. 1.4 and Sec. 1.5 present case studies for experimental evaluations. The conclusion and future work are in Sec. 1.6.

## 1.2 Related Work

Several performance modeling works were proposed as follows. For scientific workflows, the following works were proposed; for example, on a CPU node [37], in the Grid environment [11][16], and in the cloud environment [26][25]. However, the large scientific workflows are frequently running on a large computer with sophisticated storage and networking resources that are not easily captured by the existing models. Williams et al. [37] proposed the Roofline model about a theoretical model for analyzing upper bounds of performance with given computational bottlenecks and memory bottlenecks. Tikir et al. [35] proposed to use genetic algorithms to predict achievable bandwidth from cache hit rates for memory-bound HPC applications. Duan et al. [16] proposed to use a hybrid Bayesian-neural network to predict the execution time of scientific workflow in the Grid environment. In addition, performance models have been proposed in other domains. Cohen et al. [14] proposed to learn an ensemble of models using a tree-augmented Bayesian network on a system trace, and cluster the signatures to identify different regions of normality as well as recurrent performance problems. Ironmodel [33] employed a decision tree to build the performance model based on the queuing theory of expected behavior from end-to-end traces. Ganesha [28] adopted a clustering mechanism to learn the initial parameters to model Hadoop performance behavior as a mixture of *k* Gaussians. These performance models are based on the simplified models or assumptions about the executions on the underlying hardwares and cluster. Our performance analysis is based on the empirical model without sacrificing the complex interactions in the executions.

Researchers have proposed mechanisms to identify performance problems in the cluster environment. Barham et al. [5] proposed to use clustering to identify anomalous requests. Xu et al. [38] proposed to find erroneous execution paths using the PCA [19] on console logs. Bod et al. [6] used logistic regression with L1 regularization on the vector of metric quantiles to fingerprint performance crisis. They used online sampling to estimate quantiles from hundreds of machines. Vento et al. proposed to use floating point operations per seconds (flops) as an indicator of poor performance jobs [36]. Yoo et al. [41] adapted machine learning mechanisms to identify performance bottlenecks using fingerprints generated from micro-benchmarks. Yadwadkar et al. [39] proposed to use the the Support Vector Machine (SVM) [15] to proactively predict stragglers from cluster resource utilization counters. Browne et al. [8] proposed a comprehensive resource management tool by combining data from event logs, schedulers, and performance counters. In addition, Chuah et al. [13] proposed to link resource usage anomalies with system failures. These works can help our work differentiate performance bottlenecks at cluster level and those at application level. However, they also lack support to analyze large size logs from scientific workflows.

Several mechanisms have been proposed to find the causes of performance bottlenecks. Chen et al. [12] proposed to use change point detection on the latency of TCP request using conventional statistical mechanism, CUSUM and BCP. It built a causal graph using pc-algorithm. Kim et al. [21] proposed to periodically generate service dependencies and rank root cause candidates using conditional clustering. Killian et al. [20] proposed to find performance affecting changes (per-machine differences) in logs. T-tests were used to compare the two distributions and determine whether the observed differences of variances are significantly different. Sambasivan et al. [30] proposed a mechanism to diagnose the causes of performance changes in a distributed system by extending call path analysis to request flow analysis. They claim that it can find performance affecting changes in flows by comparing to the previous runtime traces. While these proposed mechanisms were not designed to analyze large size of data, they can complement our work by providing automation to identify data dependency of performance bottlenecks.

Yuan et al. [42] used signatures constructed from n-grams of system-call invocations observed during a problem occurrence. They used the Support Vector Machine (SVM) [15] to detect whether a new signature is representative of a previously observed problem. It builds a regression tree showing the low-level parameters such as function parameters, configuration values, or client-sent parameters that best separate requests in these categories. Oliner et al. [27] proposed to identify correlations in anomalies across components. Their mechanism calculates anomaly scores for discrete time-intervals by comparing the distribution of some counters such as average latency. Attariyan et al. [4] proposed performance cost evaluation using information flow analysis. Kundu et al. [22] presented performance modeling of VM-hosted applications as resource allocation and contention using machine learning mechanisms.

Several performance tools have been proposed to improve the performance of HPC applications. Shende et al. [31] designed Tau to support monitoring parallel applications by automatically inserting instrumentation routines. Böhme et al. [7] presented an automatic mechanism which performs instrumentation during compilation in order to identify the causes of waiting periods for MPI applications. Burtscher et al. [10] designed Perfexpert to automate identifying the performance bottlenecks of HPC applications with predefined rules. Adhianto et al. [3] designed HPCToolkit to measure hardware events and to correlate the events with source code to identify performance bottlenecks of parallel applications. The detection mechanisms of these tools were heavily dependent on manually created metrics and rules. Vampir [9] uses MPI workers to parallelize performance analysis computations. However, it lacks supporting distributing the computations to multiple nodes. These performance tools lack distributing and parallelizing the computations of the analysis to large number of machines. Some tools such as Tau [31] and Vampir [9] can parallelize computational loads MPI processes, and potentially these MPI processes can be extended to distribute multiple loads. However, this extension involves significant implementation challenges due to synchronization and inter-process communication complexities and lack of fault tolerance support. Instead, PATHA can interactively analyze the large size application and system logs of scientific workflows requiring large computation within user-tolerable latency. Furthermore, PATHA can complement these tools by providing mechanisms to distribute and parallelize the computational loads in addition to fault tolerance feature from read-only characteristics of RDDs.

Depending on job-specified resource requirements and the current system load, the queuing system may assign one or multiple nodes to the job, and the system captures performance information such as memory usage, CPU time, and elapsed time. However, such information is generally about the whole job, and more fine-grained information would be helpful to understand the individual steps of a large parallel workflow. Alternatively, the workflow management system could record the performance information of each step of a workflow [24], a profiler may be used to automatically capture detailed performance information [31], or the user may instrument selected operations with some library functions [34]. In these cases, the performance data is typically captured into log files. Our tool leverages these measuring mechanisms for the performance analysis.

## 1.3 Design and Implementation

PATHA is implemented over a big-data processing framework, Apache Spark$^{TM}$ [43] that distributes and parallelizes computational workloads at the parser and the analyzer levels. The PATHA supports:

- execution time analysis to find performance bottlenecks and time consuming routines in applications
- data dependency analysis to identify the possible causes of performance bottlenecks
- interactive visualization synched with performance measurements

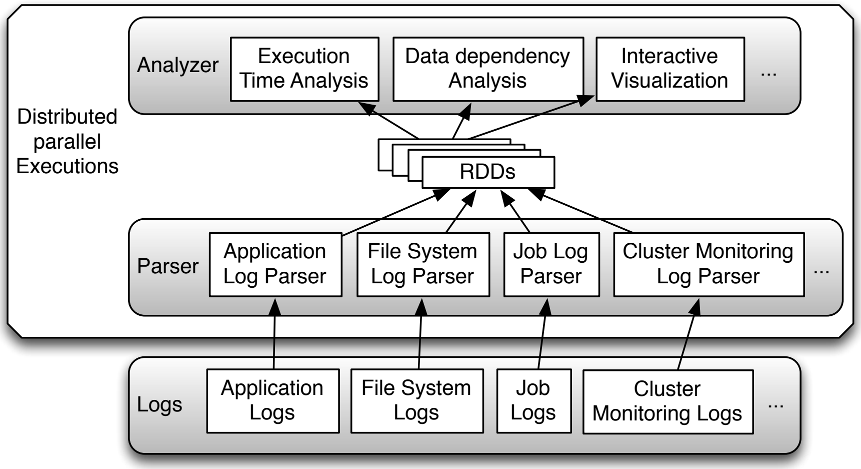


Fig. 1.1: The overview of PATHA.

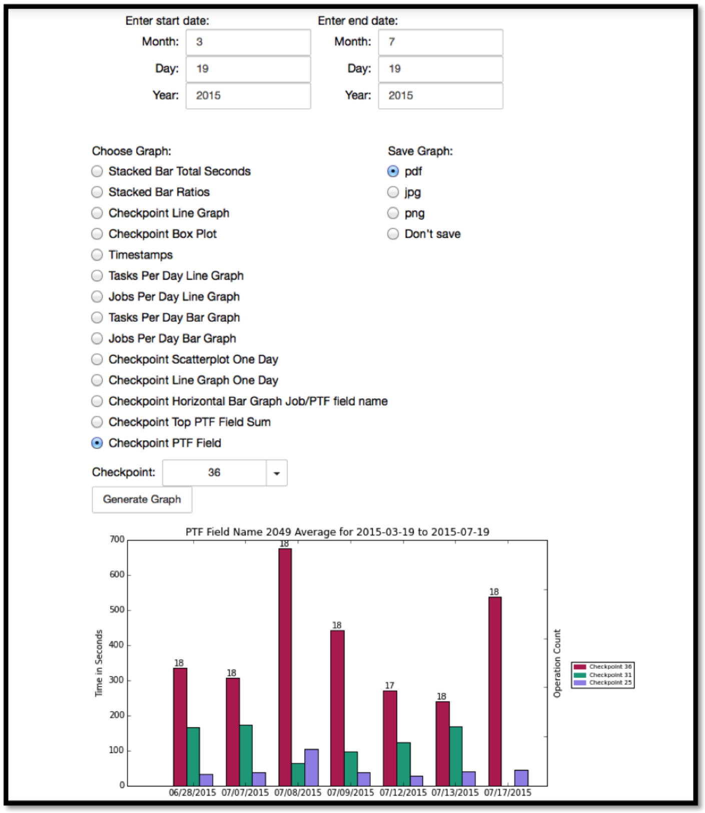
Using PATHA, performance analyses can be conducted on different types of logs and measurements in scientific clusters in addition to application logs. As shown in Fig. 1.1, each parser is implemented to parse different types of logs such as application logs, file system logs, job logs, and cluster monitoring logs. At the parser level, the different types of logs stored in parallel file system or database can be loaded into distributed memory of the multiple nodes, as a form of Resilient Distributed Datasets (RDDs). RDDs are the partitioned fault-tolerant (immutable) collection of elements that can be operated in a distributed and parallel manner on Apache Spark$^{\text{TM}}$. The computations of RDDs for parsing and loading multiple files or separate partitions in each file are distributed and computed in parallel in multiple cores and multiple nodes. Then, these parsed results are loaded into memories in multiple nodes or saved in multiple files. These loaded RDDs from the different types of logs can be analyzed separately or together in PATHA. PATHA provides the components of execution time analysis, data dependency performance analysis, and interactive visualization framework. It provides the predefined set of functions to enable users to conduct the performance analysis.

RDDs loaded as a form of rows of tuples can be computed in parallel by using the functional programming operators such as map, reduce, ‘group by key’, or ‘and sort by key’. The executions are implemented by combining these functional programming operators. In addition, computations between RDDs such as join are supported so that different types of logs can be analyzed in a combined way. This enables discovering uncovered performance issues that were difficult to be identified when the logs are separately analyzed. Users can interactively conduct performance analysis with either querying results or generating graphs by combining with grouping, aggregation, and filtering operations with the interesting fields or variables. This is to pinpoint the bottleneck locations from the execution time analysis and to identify the most significant field related to the discovered bottleneck from the data dependency analysis. In addition, it provides the platform that users can use the existing libraries of machine learning and statistics in popular programming languages, such as Java and Python, so that they can easily conduct feature selection, clustering, classification, or regression analysis. Not only conducting the PATHA-provided predefined performance analysis, but users also can implement their customized analysis by combining libraries on the loaded RDDs without consuming much time on the implementation of distributed and parallel programming.

The computations at the analyzer level are distributed and computed in parallel on multiple cores and multiple nodes similarly at the parser level. Apache Spark$^{\text{TM}}$ can be deployed in a separate cluster with several hundred nodes so that users can interactively execute analyses after connecting to the cluster. [1] While there is cost of loading data from disk and distributing data into different nodes, the analysis can

[1] The current version of Apache Spark$^{\text{TM}}$ is optimized when using local disk as an intermediate data storage instead of accessing data from a parallel file system in scientific clusters. However, the lack of local disk in scientific clusters did not impact much on performance. This was because the most of the performance analyses in PATHA were compute bound as the most of the data movement was happened in parsing and loading time.

be conducted on the loaded data from the memory. PATHA utilizes this memory cache of data as much as possible so that the loading overhead can be minimized. The crucial point is that underlying parallel execution of PATHA is dispatched in multiple nodes and multiple cores in each node without the user intervention. Therefore, PATHA can handle the large amount of different types of performance logs and measurements.



Enter start date:
Month: 3
Day: 19
Year: 2015

Enter end date:
Month: 7
Day: 19
Year: 2015

Choose Graph:
○ Stacked Bar Total Seconds
○ Stacked Bar Ratios
○ Checkpoint Line Graph
○ Checkpoint Box Plot
○ Timestamps
○ Tasks Per Day Line Graph
○ Jobs Per Day Line Graph
○ Tasks Per Day Bar Graph
○ Jobs Per Day Bar Graph
○ Checkpoint Scatterplot One Day
○ Checkpoint Line Graph One Day
○ Checkpoint Horizontal Bar Graph Job/PTF field name
○ Checkpoint Top PTF Field Sum
● Checkpoint PTF Field

Save Graph:
● pdf
○ jpg
○ png
○ Don't save

Checkpoint: 36

Fig. 1.2: The interactive visualization framework.

The performance analyses in Sec. 1.4 and Sec. 1.5 were conducted using the visualization tools and figure outputs of PATHA shown in Fig. 1.2. The interactive framework is implemented with iPython and web browser, which allows users to integrate performance analysis with the web browser front-end. As it uses web browser as front-end, the requirement of installation is much reduced, users can interactively conduct performance analysis by creating the different types of plots with different time window or conditions. The computations on data as RDDs behind this interactive analysis are distributed and executed in parallel. In addition, users can conduct execution time analysis by querying different types of graphs such as histograms, bar graphs, box plots and scatter plots. This analysis framework not only allows users to uncover performance bottlenecks in terms of execution time, but also allows them to further query and to study the possible sources of additional performance bottlenecks related to the data dependency. The case study of the PTF application shows the steps and procedures to use PATHA, and we currently work on to release PATHA as a software package along with the instructional manual. We believe that these example of use cases can generally applicable to other performance analyses for systems and applications. We plan to conduct user studies to improve the user interface and to reduce the learning curve on using PATHA. The development of this tool will continue to advance the future research of characterizing performance behavior and building performance model.

## 1.4 Case Study - PTF Application

The evaluations of PATHA using the Apache Spark™ [43] were conducted on a part of the NERSC Edison clusters with several hundred machines with two 8-core CPUs, Intel® Xeon® E5-2670 and 64 GB memory. The Palomar Transient Factory (PTF) application was used as a case study to evaluate PATHA. We have used the PTF application logs collected on the NERSC Edison cluster system from Mar. 19, 2015 to Jul. 18, 2015 (PST). The PTF application was executed on the compute nodes of the NERSC Edison cluster system assigned for regular applications with two 12-core CPUs, Intel® Xeon® E5-2695 and 64 GB memory. [2] We used Apache Spark™ to distribute and parallelize computational loads for PATHA. PATHA allowed more thorough investigation on the PTF application measurements and derived values from the measurements such as the average execution time by averaging differences of the measured timestamps in multiple tasks in each job. While we were able to select the number of machines up to several hundreds for our experiments, the executions were not exclusive in the allocated machines. We plan to set up exclusive executions on the allocated machines to analyze the scalability of PATHA. Due to our effort to reduce the overhead of data loading and distribution, our initial observations confirmed that PATHA was scalable in several hundred machines without degradation even with the interferences from other job executions.

### *1.4.1 PTF Application*

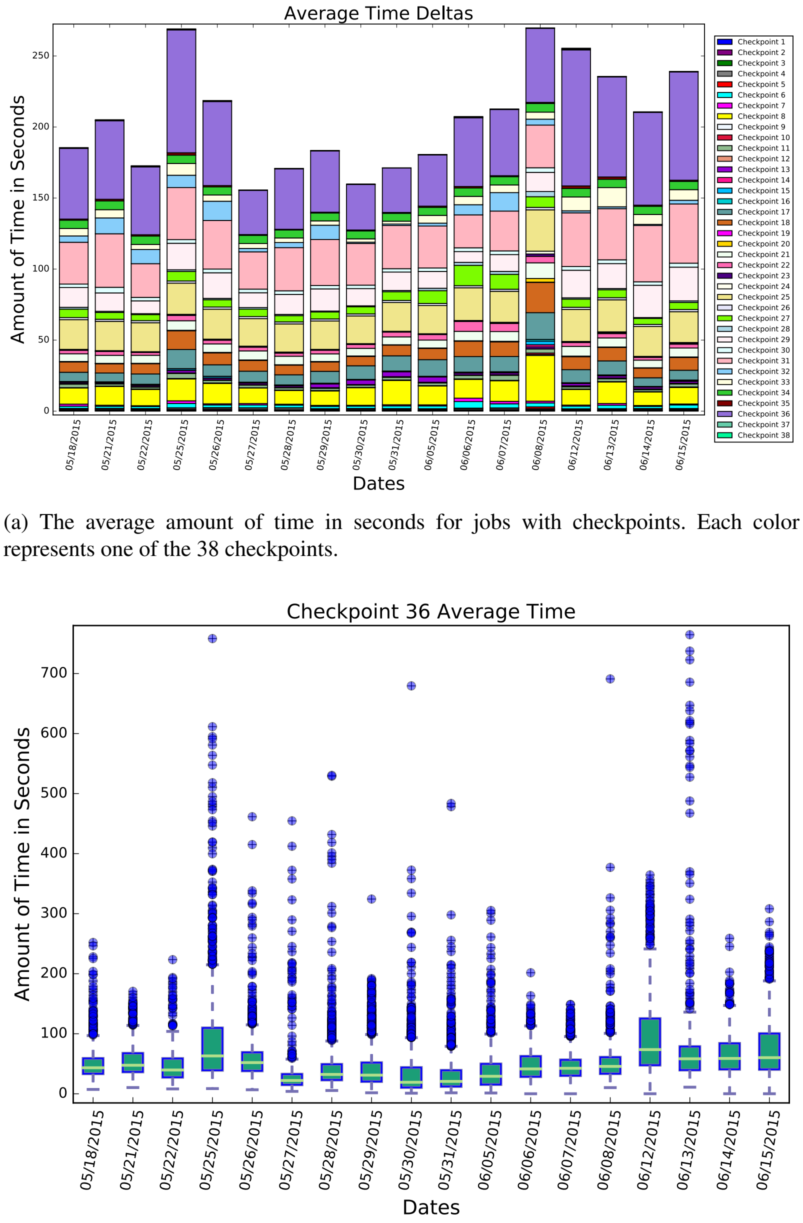
The PTF application focuses on expanding our knowledge of transient phenomena such as supernova explosions and massive star eruptions [23]. There are four large-scale photometric and spectroscopic surveys that generate and/or utilize hundreds of gigabytes of data per day, and the PTF application is one of them. The transient

[2] The Edison cluster system for PATHA has different configurations with that of the Edison cluster system for the PTF.

detection survey component of the PTF has been performed at the automated Palomar Samuel Oschin 48-inch Schmidt telescope equipped with a camera that covers a sky area of 7.3 square degrees in a single snapshot. Data taken with the camera are transferred to NERSC Edison where running a real-time reduction pipeline. The pipeline matches images taken at different nights under different observing conditions and performs image subtraction to search for transients. The transient candidates out of this pipeline then pass through machine-learning classifiers to be prioritized for real transients over artifacts. The final output is then displayed through a web portal for visual inspection by human. This pipeline has achieved the goal of identifying optical transients within minutes of images being taken.

For the case study with the PTF application, we used its measurement logs that were collected in the database. The size of entire database is 1.6 TB. Timestamps, job id, task id, and checkpoint id were loaded into RDDs for execution time analysis. The execution time at each checkpoint was computed for each job and task. Then, the execution times were grouped by different keys, e.g., job or task, and the average execution times were computed with the keys. For this grouping, RDDs were needed to include columns with distinctive values to be used as keys such as job id, task id, and checkpoint id. During computation for the average, missing timestamps or unordered timestamps were filtered out. These irregularities were caused by various reasons, e.g., failures in the executions at application level or system level. To implement filtering out these using database query or customized user application can be challenging and costly. For data dependency performance analysis, the execution times were computed with multiple associated variables or fields that were potentially related to the identified performance bottlenecks.

Fig. 1.3 depicts the average amount of time in seconds that the PTF analysis pipeline took on each day to execute all jobs and tasks, which were executed from May 18, 2015 to June 15, 2015. The execution of PTF application involves the executions of multiple jobs computing different areas. Each job consists of 10 tasks whose checkpoints are stored in database when each processing step is conducted. As shown in Fig. 1.3(a), the PTF analysis pipeline consists of 38 checkpoints, with each color representing a different checkpoint. The top five checkpoints with the longest execution time taken over a span of 64 days were checkpoints 8, 25, 29, 31, and 36 in Fig. 1.3(a). The average daily percentage calculations revealed that checkpoint 8 took on average 7.29%, checkpoint 25 takes 11.16%, checkpoint 29 takes 6.22%, checkpoint 31 takes 14.79%, and most notably, checkpoint 36 takes 23.72% on average. The three checkpoints with the longest average execution times were further investigated for a potential bottleneck where performance could be improved.



(a) The average amount of time in seconds for jobs with checkpoints. Each color represents one of the 38 checkpoints.

(b) The amount of time in seconds for jobs with checkpoint 36, where each vertical line is for one day. The line in the middle of a box marks the median time, the brackets of a box mark the interquartile ranges (IQRs), the high whisker is at $Q3 + 1.5 \times IQR$, and the circles mark the instances with extra long execution time.

Fig. 1.3: The average amount of time in seconds for checkpoint operations, which was executed from May 18,2015 to June 15,2015.

### *1.4.2 Execution Time Analysis*

Using PATHA, we conducted execution time analysis on checkpoint 36 specifically. The Transients in the Local Universe (TILU) query - a geometric query that correlates the incoming candidates with the table of known galaxies with their elliptical shapes and orientations. Fig. 1.3(b) shows the box plot of average execution time of this query together with the performance outliers as circles. We see that many jobs took much longer than the average time. Based on this observation, we focused on certain days, such as June 13, 2015 that has larger variance and many outliers. Further study about this day will be presented next paragraphs.

Fig. 1.4(a) shows the scatter plot of the amount of time in seconds for each job throughout the day starting at approximately 04:30, when the first task of checkpoint 25, 31, and 36 was executed on June 13, 2015. Fig. 1.4(b) shows the scatter plot for checkpoint 36, which shows the spikes of an execution time during the time period from 08:20 to 08:45 on June 13, 2015. This particular time window would need further investigation.

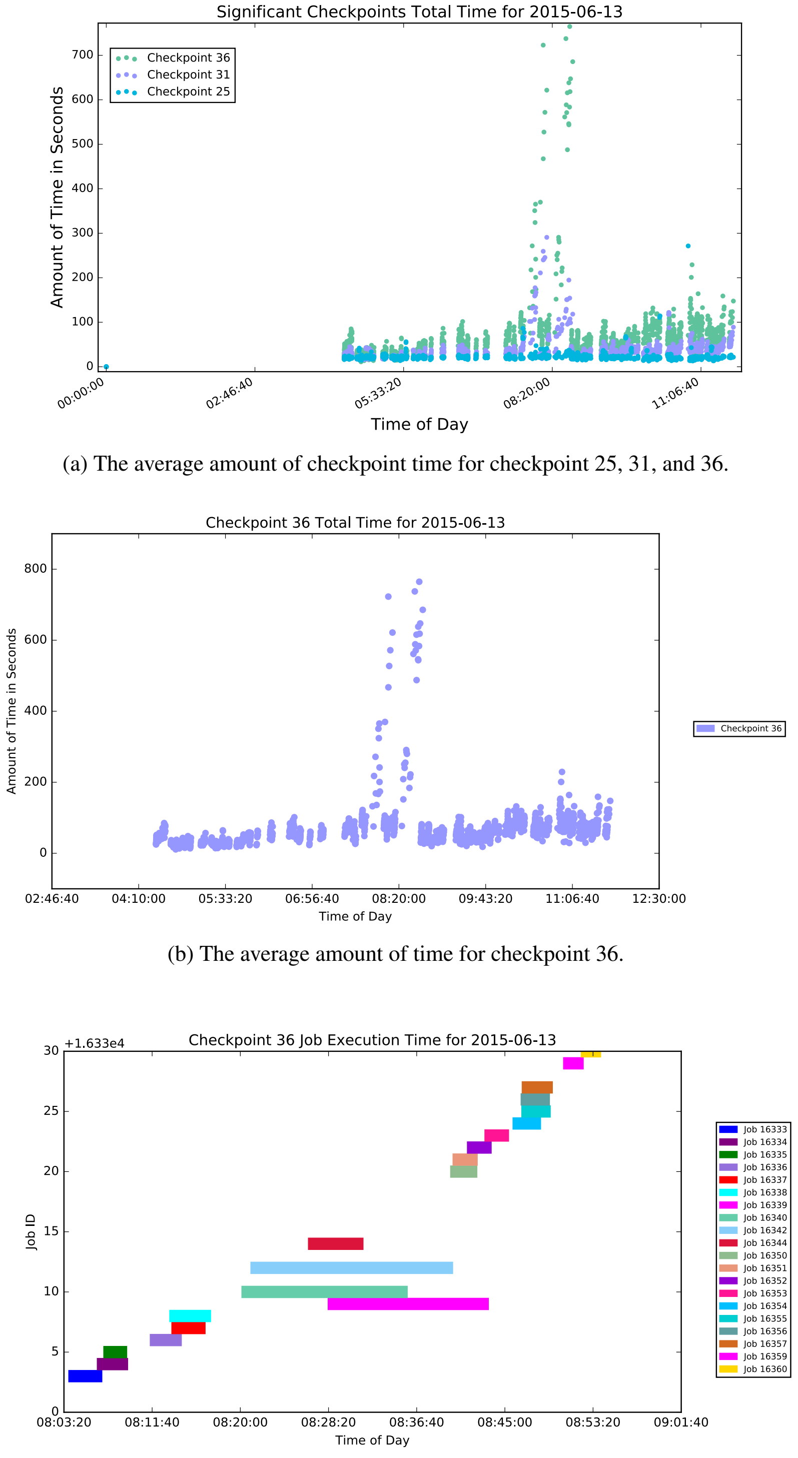
Fig. 1.4(c) shows the time spent by each instance of TILU query in the time window from 08:20 to 08:45. By focusing on the executions in this specific time duration with significantly higher execution times, we can discern whether bottlenecks are caused by cluster load competing system resources or caused by application-specific reasons. The length of each bar in Fig. 1.4(c) shows the total execution time of each job, and its corresponding job IDs for the checkpoint operation 36. The jobs with longest execution time had job IDs 16339, 16340, and 16342. Interestingly, the other job, 16353 that was executed in the similar time window showed much smaller execution times. These instances of long execution time were interspersed with normal looking instance showing much smaller execution time. Therefore, we speculate that system loads due to competing for shared resources did not cause their long execution times. Additional possibility would be studied in the next section about whether these long execution times had data dependencies in user data.

### *1.4.3 Data Dependency Performance Analysis*

We studied two attributes in the PTF application to see how they affected the execution time and how the performance of the PTF application depended on them, based on the suggestions from application scientists. These attributes were: the number of saved objects and the galactic latitude.

#### **Analysis of Saved Objects**

In the PTF application, a fragmentation algorithm is performed on the subtraction images to identify variable stars and transient candidates over the noisy background

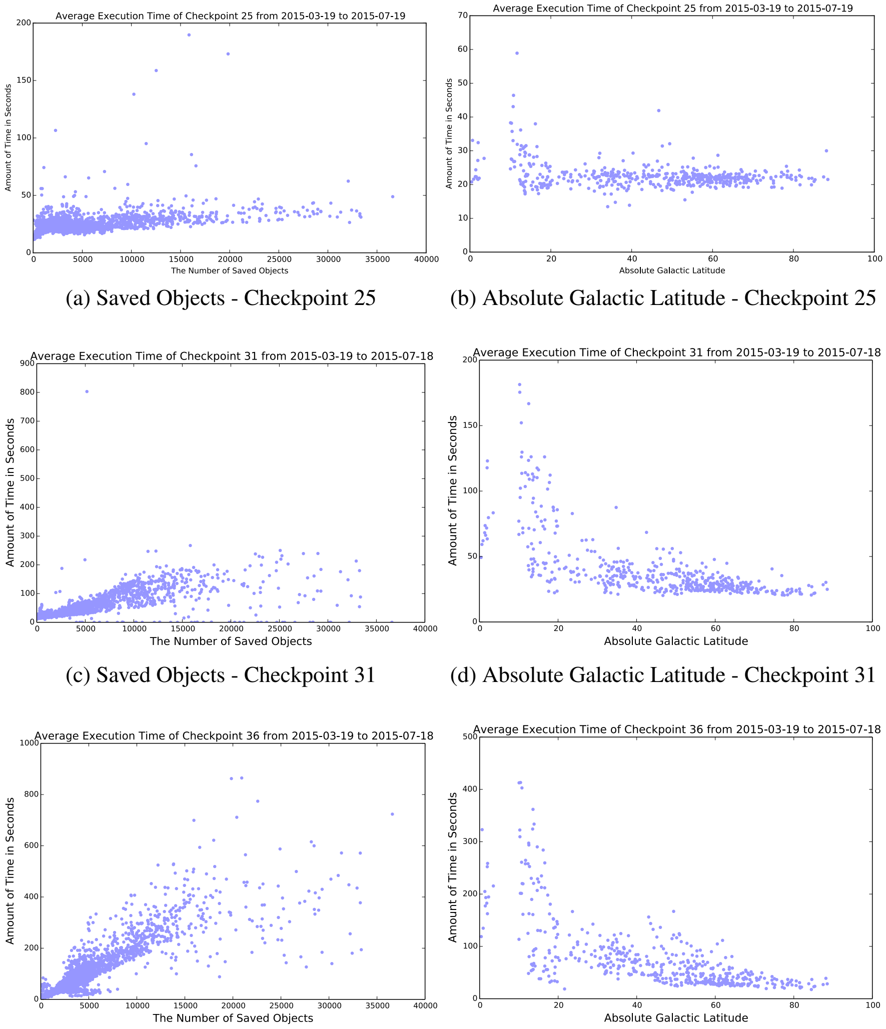


(a) The average amount of checkpoint time for checkpoint 25, 31, and 36.

(b) The average amount of time for checkpoint 36.

(c) The execution times of all jobs with their corresponding job IDs during the time period 12:20 to 13:06 on June 13, 2015.

Fig. 1.4: The amount of time in seconds per day for each job on June 13, 2015.



(a) Saved Objects - Checkpoint 25 (b) Absolute Galactic Latitude - Checkpoint 25

(c) Saved Objects - Checkpoint 31 (d) Absolute Galactic Latitude - Checkpoint 31

(e) Saved Objects - Checkpoint 36 (f) Absolute Galactic Latitude - Checkpoint 36

Fig. 1.5: The average execution time of checkpoints 25, 31, and 36 for number of saved objects and absolute galactic latitude.

and to measure their shape parameters such as the length and angle of its major axis and ellipticity. Then, a simple shape cut is applied to remove elongated candidates that are probably artifacts. The candidates that pass the shape cut are saved for further examination, i.e., checkpoints after the checkpoint 25. The reason of having different numbers of saved objects is that the total number of candidates for further examination is determined by the number of variable stars (since real transients are rare), which in turn correlates with the total number of stars in a given field. Fig. 1.5(a, c, e) show the average execution time of checkpoints 25, 31 and 36 for the number of saved objects, and the linear relation between the average execution time and the number of saved objects. [3] It shows the performance bottleneck in these checkpoints when computed with the large number of stored objects. This is because the large number of saved objects requires more comparisons and computation. This identified bottleneck would lead to reduce the computation time when computing with the large number of stored objects.

#### Analysis of Galactic Latitude

Fig. 1.5(b, d, f) illustrate a correlation between the execution times of three checkpoints (25, 31, and 36) and the absolute galactic latitude (zero degree corresponds to the Milky Way plane), and the performance bottlenecks is shown at low galactic latitudes. The physical reason behind it is that the closer a field is to the Milky Way, the more celestial objects, the more transient/variable candidates, and the longer execution time for these checkpoints. At low galactic latitudes, i.e., close to the Milky Way plane, the stellar density is higher, and so is the density of variable stars. Therefore, images taken at low galactic latitudes in general generate more candidates than those at high galactic latitudes.

With the identified data dependencies, we optimized the application pipeline for the checkpoint 31, where we parallelized the most time consuming routines when computing the larger number of saved objects and at low absolute galactic latitudes. The optimization showed the reduced execution time up to 2.05 times. We can further improve overall performance of the checkpoint 31 by applying the parallel executions more intelligently. Instead of making all executions in parallel including small execution times, we can only make executions in parallel that supposedly take much larger execution time with larger number of saved objects and at low absolute galactic latitudes. For this purpose, we plan to analyze how to adjust parallelism depending on the number of saved objects and absolute galactic latitudes.

## 1.5 Case Study - Job Log Analysis

### *1.5.1 Job Logs*

Scientific clusters generally consist of a job scheduling engine, compute nodes for assigned jobs, storage nodes for a parallel file system, data transfer nodes for network accesses, and special purpose nodes for database or web services. Scientific clusters contain sufficiently large number of nodes and multiple parallel executions

[3] The linear regression coefficients are $5.673 \times 10^{-3}$ for checkpoint 31 and $8.515 \times 10^{-4}$ for checkpoint 36.

from the tasks that incur complexity challenges for analysis from developers and system administrators. Due to the complexity and the size of the job logs, it is challenging for developers and system administrators to analyze and extract meaningful information from the logs. For example, one can attempt to select jobs with the top-most resource usage to analyze whether they experience performance anomalies, and this task can be a challenge because of the large amount of job logs and the concurrent data accesses on the shared resources.

### *1.5.2 Test Setup*

In our experiments, we have used the job logs collected on the Genepool cluster at NERSC, consisting of 774 nodes [1]. The logs were written by the Univa Grid Engine [2] when each job was finished. The size of the logs from the Genepool cluster from July 29, 2014 to Feb. 28, 2016 was 4.5 TB that can incur significant computational challenges for an analysis. To generate plots in later sections, we have used the part of the job logs from Jan. 1, 2015 to Jan. 31, 2015 (GMT) with 2.4 million records or 1.1 GB. It contains 45 fields such as host name, job name, failed code, exit code, and resource usages. We selected 13 performance-related fields: wall clock, user/system CPU time, soft/hard page faults, file block input/output, voluntary/involuntary context switches, aggregated memory usage, aggregated I/O, maximum resident set size, and maximum virtual memory. Tab. 1.1 describes these fields.

Table 1.1: The description of performance-related fields.

| Feature | Description |
|---|---|
| Wall clock | The duration between start and end of a task |
| User CPU time | The sum of time spent on CPU cores at the user level |
| System CPU time | The sum of time spent on CPU cores at the system level |
| CPU time | The sum of the user CPU time and system CPU time |
| Maximum resident set size | Maximum value of utilized memory size during job execution |
| Page reclaims | Soft page faults without involving I/O |
| Page faults | Hard page faults with involving I/O |
| Block input operations | The number of times that the file system had to perform input |
| Block output operations | The number of times that the file system had to perform output |
| Voluntary context switches | The number of times for voluntary context switches |
| Involuntary context switches | The number of times for involuntary context switches |
| Memory | The integral memory usage in Gbytes $*$ CPU time in seconds |
| IO | The amount of data transferred in input/output operations |

### *1.5.3 Job Log Analysis*

Scientific clusters have encountered technical advances that involve increasing scales of data volumes, number of machines, and exploited parallelism in software executions. This leads to unforeseen scales of interactions in software executions between hardware components and nodes. Developers often encounter difficulties to gather information about the details of underlying hardwares and runtime information of a cluster containing large number of nodes. On the other hand, system administrators are overwhelmed by large-scale performance-related logs and noises from interactions and interferences in the executions of a cluster. Due to these reasons, it is challenging to analyze system performance on scientific clusters. For these challenges, PATHA provides big-data-ready performance analysis features with much less developmental costs.

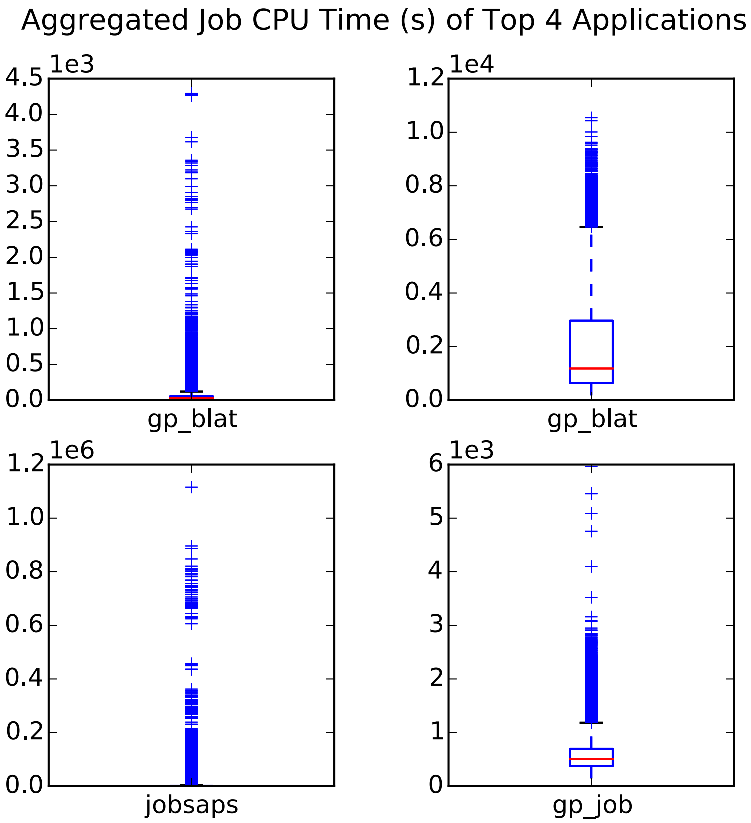


Fig. 1.6: The aggregated CPU time (s) from top 4 applications

Fig. 1.6 shows the aggregated CPU time(s) from top 4 frequently executed applications on Genepool cluster at NERSC between Jan. 1, 2015 and Jan. 31, 2015 (GMT). The most frequently executed applications need to be selected. Then the job executions from these selected top jobs (top 4 jobs in 1.6) need to be selected. CPU times of each job execution also need to be aggregated as one job can have multiple sub-job (task) executions. The aggregated CPU times can be plotted as box plots, and application developers can analyze whether the executed jobs spend unexpectedly large CPU time. In addition, the frequently executed top applications can be analyzed for expected performance, and PATHA makes these analyses easier.

### *1.5.4 Clustering Analysis*

Clustering analysis groups task executions represented as points of features sharing similar performance characteristics. In our case, the features correspond to the selected fields of the job logs. The points as the target of clustering correspond to each record of the job logs representing a completed execution. The scales of the features differ by multiple orders of magnitude as they show the different types of fields of the job logs. The scale of each field is adjusted by $L1$-*norm* scaling of each field.

We applied K-means clustering algorithm [17], which makes clusters with a specified `k` as a number of clusters. Using clustering analysis, we can identify the minor clusters containing small number of points, and these minor clusters have different characteristics from other major clusters containing large number of points. When the centroids of these minor clusters are separated from the centroid of the major clusters with significant distances, this can mean that the minor clusters have significantly different characteristics. Therefore, these minor clusters can be good targets for further performance analysis where significantly different characteristics are resulted from.

Table 1.2: The size of clustering results with different scalings and $k$: $C_i$ is the $i$th cluster.

| $k$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ |
|---|---|---|---|---|---|---|---|---|
| 2 | 79 | 2.40M | | | | | | |
| 4 | 1 | 67 | 78 | 2.4M | | | | |
| 8 | 1 | 1 | 22 | 78 | 78 | 115 | 1456 | 2.4M |

Tab. 1.2 shows the size of clustering results with different $k$. For instance, in Tab. 1.2 with $L1$-*norm* scaling, $C_1$ (size 79) with $k = 2$ is divided into $C_1$ (size 1) and $C_3$ (size 78) with $k = 4$. In addition, $C_1$ and $C_3$ with $k = 4$ are repeatedly identified with the same size in $C_1$ and $C_4$ wit $k = 8$. We inspected these two clusters, and discovered that they contain extraordinarily high values of integrated memory usage above the theoretically possible value. We speculated that this was a system glitch

and required further investigation of its causes. We believe that this example shows the usage of the tool towards the identification of the performance outliers from the clustering results.

Plotting results from the clustering algorithm help validate fitting results. As the size of features is 13, it is not possible to directly plot on 2D or 3D space. To enable plotting, we applied dimensionality reduction method, PCA [19]. Fig. 1.7 shows that the PCA-transformed plot of clustering results for the original $L1$-*norm* scaled data in Tab. 1.2. Each cluster is colored in a different color. The centers of the clusters are represented with cross marks with the same color of the cluster. The points in clusters are randomly selected in log-log scale of the size of the clusters. This is to reduce the number of points that are most likely redundant in large clusters. However, this selection may cause an underestimation of large clusters. In addition, the sizes of cross marks of centers are increased in log-log scale to represent the size of clusters. Fig. 1.7 shows the identified performance outliers. Fig. 1.7(a, b, c) show significantly distant minor clusters near (0.1,0), (0.2,0), and (0.4,0). They were the clusters, $C_1$ (size 79) with $k = 2$ and $C_1$ (size 1) and $C_3$ (size 78) with $k = 4$ in Tab. 1.2. [4]

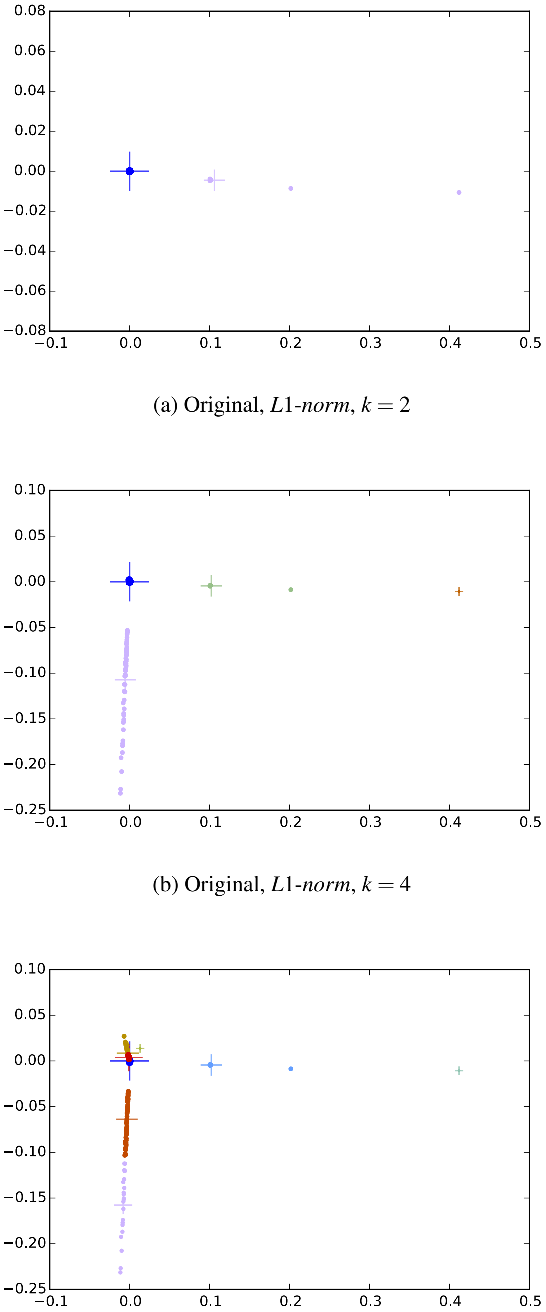
## 1.6 Conclusion

As the computations and analyses of large datasets are distributed and parallelized on multiple compute-nodes, it becomes challenging to analyze the performance issues related to the applications and hardware platforms due to the large collection of performance measurements. In order to tackle this challenge, we have developed PATHA (Performance Analysis Tool for HPC Applications) using an open-source big data processing framework. The HPC applications as referred to here include parallel applications, high-throughput computing applications, and other applications for Big Data processing.

Users can use PATHA to identify performance characteristics and performance bottlenecks in their science applications and scientific clusters. PATHA can analyze large volume of performance measurement data from large science projects. It provides the execution time analysis to find performance bottlenecks and time-consuming routines in applications, data dependency performance analysis to identify possible data dependencies of discovered performance bottlenecks. These analyses can be interactively conducted by using different types of performance measurements from scientific clusters.

We have conducted two case studies to evaluate PATHA. With the PTF application, we identified performance bottlenecks in checkpoint operations 25, 31, and 36. We also identified their direct data dependencies on the number of saved objects and the absolute galactic latitude. Developers of the PTF application have been working on optimizing identified performance bottlenecks, and the execution time

[4] Please note that the points in Fig. 1.7(b) and Fig. 1.7(c) near (0,[-0.05, -0.25]) are not shown in Fig. 1.7(a) as they are the part of the major cluster near (0,0).



(a) Original, $L1$-*norm*, $k = 2$

(b) Original, $L1$-*norm*, $k = 4$

(c) Original, $L1$-*norm*, $k = 8$

Fig. 1.7: PCA-transformed clustering results

has been reduced up to 2.05 times. In the other case study with the job logs, we have analyzed system performance in a large scientific cluster. We were also able to identify system performance outliers using clustering analysis and dimensionality reduction method.

For the future work, we plan to use the PATHA in the extended analysis combining the measurements of hardware executions in scientific clusters and the measurements from the applications. In addition, we plan to automate the process of bottleneck identification. These will help identify the performance bottlenecks due to the system related issues along with the application related issues.

## 1.7 Acknowledgments

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, the U.S. Dept. of Energy, under Contract No. DE-AC02-05CH11231. This work used resources of NERSC. The authors would like to thank Douglas Jacobson, Jay Srinivasan, and Richard Gerber at NERSC, Bryce Foster and Alex Copeland at JGI, and Arie Shoshani at LBNL.

## References

1. Genepool cluster. `http://www.nersc.gov/users/computational-systems/genepool` (2015)
2. Univa grid engine. `http://www.univa.com/products/grid-engine.php` (2015)
3. Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., Tallent, N.R.: HPCTOOLKIT: tools for performance analysis of optimized parallel programs. Concurr. Comput. : Pract. Exper. **22**(6), 685–701 (2010)
4. Attariyan, M., Chow, M., Flinn, J.: X-ray: Automating Root-Cause Diagnosis of Performance Anomalies in Production Software. In: OSDI '12: Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, pp. 307–320 (2012)
5. Barham, P., Donnelly, A., Isaacs, R., Mortier, R.: Using magpie for request extraction and workload modelling. In: OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, pp. 259–272. USENIX (2004)
6. Bod, P., Berkeley, U.C., Goldszmidt, M., Fox, A., Berkeley, U.C., Woodard, D.B., Andersen, H., Bodik, P., Goldszmidt, M., Fox, A., Woodard, D.B., Andersen, H.: Fingerprinting the datacenter. In: EuroSys'10: Proceedings of the 5th European conference on Computer systems, pp. 111–124. ACM, New York, New York, USA (2010). DOI 10.1145/1755913.1755926
7. Bohme, D., Geimer, M., Wolf, F., Arnold, L.: Identifying the Root Causes of Wait States in Large-Scale Parallel Applications. In: Proceedings of the 2010 39th International Conference on Parallel Processing, pp. 90–100. IEEE (2010)
8. Browne, J.C., DeLeon, R.L., Lu, C.D., Jones, M.D., Gallo, S.M., Ghadersohi, A., Patra, A.K., Barth, W.L., Hammond, J., Furlani, T.R., McLay, R.T.: Enabling comprehensive data-driven system management for large computational facilities. In: High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for, pp. 1–11. DOI 10.1145/2503210.2503230

9. Brunst, H., Winkler, M., Nagel, W.E., Hoppe, H.C.: Performance optimization for large scale computing: The scalable vampir approach. In: Computational Science-ICCS 2001, pp. 751–760. Springer (2001)
10. Burtscher, M., Kim, B.D., Diamond, J., McCalpin, J., Koesterke, L., Browne, J.: PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–11 (2010)
11. Cao, J., Kerbyson, D., Papaefstathiou, E., Nudd, G.R.: Performance modeling of parallel and distributed computing using pace. In: Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the IEEE International, pp. 485–492 (2000). DOI 10.1109/PCCC.2000.830354
12. Chen, P., Qi, Y., Zheng, P., Hou, D.: CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In: INFOCOM'14: Proceedings IEEE International Conference of Computer Communications, pp. 1887–1895. IEEE (2014). DOI 10.1109/INFOCOM.2014.6848128
13. Chuah, E., Jhumka, A., Narasimhamurthy, S., Hammond, J., Browne, J.C., Barth, B.: Linking resource usage anomalies with system failures from cluster log data. In: 2013 IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS), pp. 111–120. DOI 10.1109/SRDS.2013.20
14. Cohen, I., Chase, J.S., Goldszmidt, M., Kelly, T., Symons, J.: Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In: OSDI, vol. 6, pp. 231–244. USENIX (2004)
15. Cortes, C., Vapnik, V.: Support-vector networks. Mach Learn **20**(3), 273–297 (1995). DOI 10.1007/BF00994018
16. Duan, R., Nadeem, F., Wang, J., Zhang, Y., Prodan, R., Fahringer, T.: A hybrid intelligent method for performance modeling and prediction of workflow activities in grids. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, pp. 339–347. IEEE Computer Society, Washington, DC, USA (2009). DOI 10.1109/CCGRID.2009.58
17. Hartigan, J.A., Wong, M.A.: Algorithm AS 136: A K-Means Clustering Algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics) **28**(1), 100–108 (1979). DOI 10.2307/2346830
18. Hey, T., Tansley, S., Tolle, K. (eds.): The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft (2009)
19. Jolliffe, I.: Principal Component Analysis. In: Wiley StatsRef: Statistics Reference Online. John Wiley & Sons, Ltd (2014)
20. Killian, C., Nagaraj, K., Killian, C., Neville, J.: Structured comparative analysis of systems logs to diagnose performance problems. In: NSDI'12: Proceedings of the 9th USENIX conference on Networked systems design and implementation. USENIX (2012)
21. Kim, M., Sumbaly, R., Shah, S.: Root cause detection in a service-oriented architecture. ACM SIGMETRICS Performance Evaluation Review **41**(1), 93–104 (2013). DOI 10.1145/2465529.2465753
22. Kundu, S., Rangaswami, R., Gulati, A., Zhao, M., Dutta, K.: Modeling virtualized applications using machine learning techniques. ACM Sigplan Notices **47**(7), 3–14 (2012)
23. Law, N.M., Kulkarni, S.R., Dekany, R.G., Ofek, E.O., Quimby, R.M., Nugent, P.E., Surace, J., Grillmair, C.C., Bloom, J.S., Kasliwal, M.M., Bildsten, L., Brown, T., Cenko, S.B., Ciardi, D., Croner, E., Djorgovski, S.G., Eyken, J.v., Filippenko, A.V., Fox, D.B., Gal-Yam, A., Hale, D., Hamam, N., Helou, G., Henning, J., Howell, D.A., Jacobsen, J., Laher, R., Mattingly, S., McKenna, D., Pickles, A., Poznanski, D., Rahmer, G., Rau, A., Rosing, W., Shara, M., Smith, R., Starr, D., Sullivan, M., Velur, V., Walters, R., Zolkower, J.: The palomar transient factory: System overview, performance, and first results. Publications of the Astronomical Society of the Pacific **121**(886), pp. 1395–1408 (2009)
24. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M.B., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system. Concurrency and Computation: Practice and Experience **18**(10), 1039–1065 (2006)

25. Malawski, M., Juve, G., Deelman, E., Nabrzyski, J.: Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, pp. 22:1–22:11. IEEE Computer Society Press, Los Alamitos, CA, USA (2012)
26. Matsunaga, A., Fortes, J.A.B.: On the use of machine learning to predict the time and resources consumed by applications. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGRID '10, pp. 495–504. IEEE Computer Society, Washington, DC, USA (2010). DOI 10.1109/CCGRID.2010.98
27. Oliner, A.J., Kulkarni, A.V., Aiken, A.: Using correlated surprise to infer shared influence. In: DSN'10: IEEE/IFIP International Conference on Dependable Systems & Networks, pp. 191–200. IEEE (2010). DOI 10.1109/DSN.2010.5544921
28. Pan, X., Tan, J., Kavulya, S., Gandhi, R., Narasimhan, P.: Ganesha: blackBox diagnosis of MapReduce systems. ACM SIGMETRICS Performance Evaluation Review **37**(3), 8–13 (2009). DOI 10.1145/1710115.1710118
29. Rusu, F., Nugent, P., Wu, K.: Implementing the palomar transient factory real-time detection pipeline in GLADE: Results and observations. In: Databases in Networked Information Systems, *Lecture Notes in Computer Science*, vol. 8381, pp. 53–66 (2014)
30. Sambasivan, R.R., Zheng, A.X., Rosa, M.D., Krevat, E., Whitman, S., Stroucken, M., Wang, W., Xu, L., Ganger, G.R., De Rosa, M., Krevat, E., Whitman, S., Stroucken, M., Wang, W., Xu, L., Ganger, G.R.: Diagnosing performance changes by comparing request flows. In: NSDI'11: Proceedings of the 8th USENIX conference on Networked systems design and implementation. USENIX (2011)
31. Shende, S.S., Malony, A.D.: The TAU parallel performance system. International Journal of High Performance Computing Applications **20**(2), 287–311 (2006)
32. Shoshani, A., Rotem, D. (eds.): Scientific Data Management: Challenges, Technology, and Deployment. Chapman & Hall/CRC Press (2010)
33. Thereska, E., Ganger, G.R.: Ironmodel: robust performance models in the wild. ACM SIGMETRICS Performance Evaluation Review **36**(1), 253–264 (2008). DOI 10.1145/1375457.1375486
34. Tierney, B., Johnston, W., Crowley, B., Hoo, G., Brooks, C., Gunter, D.: The netlogger methodology for high performance distributed systems performance analysis. In: High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on, pp. 260–267 (1998). DOI 10.1109/HPDC.1998.709980
35. Tikir, M., Carrington, L., Strohmaier, E., Snavely, A.: A genetic algorithms approach to modeling the performance of memory-bound computations. In: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, p. 47. ACM (2007)
36. Vento, D.D., Hart, D.L., Engel, T., Kelly, R., Valent, R., Ghosh, S.S., Liu, S.: System-level monitoring of floating-point performance to improve effective system utilization. In: High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, pp. 1–6
37. Williams, S., Waterman, A., Patterson, D.: Roofline: An insightful visual performance model for multicore architectures. Commun. ACM **52**(4), 65–76 (2009). DOI 10.1145/1498765.1498785
38. Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.I.: Detecting large-scale system problems by mining console logs. In: SOSP'09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 117–131. ACM (2009). DOI 10.1145/1629575.1629587
39. Yadwadkar, N.J., Ananthanarayanan, G., Katz, R.: Wrangler: Predictable and Faster Jobs Using Fewer Resources. In: Proceedings of the ACM Symposium on Cloud Computing, SOCC '14, pp. 26:1–26:14. ACM, New York, NY, USA (2014). DOI 10.1145/2670979.2671005. URL `http://doi.acm.org/10.1145/2670979.2671005`
40. Yoo, W., Koo, M., Cao, Y., Sim, A., Nugent, P., Wu, K.: Patha: Performance analysis tool for hpc applications. In: IPCCC'15: Proceedings of the 34st IEEE International Performance Computing and Communications Conference (2015)
41. Yoo, W., Larson, K., Baugh, L., Kim, S., Campbell, R.H.: ADP: automated diagnosis of performance pathologies using hardware events. In: Proceedings of the 12th ACM

SIGMETRICS/PERFORMANCE, vol. 40, pp. 283–294. ACM, New York, New York, USA (2012). DOI 10.1145/2254756.2254791

42. Yuan, C., Lao, N., Wen, J.R., Li, J., Zhang, Z., Wang, Y.M., Ma, W.Y.: Automated known problem diagnosis with event traces. In: EuroSys'06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems, vol. 40, pp. 375–388. ACM (2006). DOI 10.1145/1218063.1217972
43. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster Computing with Working Sets. In: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10. USENIX (2010)