A Cloud Performance Analytics Framework to Support Online Performance Diagnosis and Monitoring Tools

Amitabha Banerjee†
Performance Engineering
Vmware, Inc.
3401 Hillview Ave, Palo Alto, CA
banerjeea@vmware.com

Abhishek Srivastava
Performance Engineering
Vmware Software India Pvt. Ltd.
Phase-4 JP Nagar, Bengaluru, KA, India
srivastavaab@vmware.com

ABSTRACT

Traditionally, performance analysis, debugging, triaging, troubleshooting, and optimization are left in the hands of performance experts. The main rationale behind this is that performance engineering is considered a specialized domain expertise, and therefore left to the trained hands of experts. However, this approach requires human manpower to be put behind every performance escalation. This is no longer future proof in enterprise environments because of the following reasons:

(i) Enterprise customers now expect much quicker performance troubleshooting, particularly in cloud platforms as Software As A Service (SaaS) offerings where the billing is subscription based, (ii) As products grow more distributed and complex, the number of performance metrics required to troubleshoot a performance problem implodes, making it very time consuming for human intervention and analysis, and (iii) Our past experiences show that while many customers land up on similar performance issues, the human effort to troubleshoot each of these performance issues in a different infrastructural environment is non-trivial. We believe that data analytics platforms that can quickly mine through performance data and point out potential. bottlenecks offer a good solution for non-domain experts to debug and solve a performance issue. In this work, we showcase a cloud based performance data analytics framework which can be leveraged to build tools which analyze and root-cause performance issues in enterprise systems. We describe the architecture of this framework which consists of: (i) A cloud service (which we term as a plugin), (ii) Supporting libraries that may be used to interact with this plugin from end-systems such as computer servers or appliance Virtual Machines (VMs), and (iii) A solution to monitor and analyze the results delivered by the plugin. We demonstrate how this platform can be used to develop different performance analyses and debugging tools. We provide one example of a tool that we have built on top of this framework and released: VMware Virtual SAN (vSAN) performance diagnostics.

We specifically discuss how collecting performance data in the cloud from over a thousand deployments, and then analyzing to detect performance issues, helped us write rules that can easily detect similar performance issues. Finally, we discuss a framework for monitoring the performance of the rules and improving them.

ICPE '19, April 7–11, 2019, Mumbai, India

ACM ISBN 978-1-4503-6239-9/19/04…$15.00
https://doi.org/10.1145/3297663.3309675

CCS CONCEPTS

• Software and its engineering~Software performance

KEYWORDS

Online Performance Troubleshooting

ACM Reference format:
Amitabha Banerjee and Abhishek Srivastava. 2019. A Cloud Performance Analytics Framework to support online performance diagnosis and monitoring tools. In *Proceedings of ACM/SPEC International Conference on Performance Engineering (ICPE'19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 8 pages. DOI: https://doi.org/10.1145/3297663.3309675

1 Introduction

1.1 Current state of the art

There exists a wide variety of research literature describing techniques to detect, triage, and fix performance issues. [7] has a comprehensive summary of the state of art of different techniques and software packages for such efforts. However, the main challenge is to identify the right technique to use for the type of performance issue under investigation. Hence the art of investigating performance issues is limited to a few individuals who work in a highly decentralized manner to tackle the problem at hand.

On the other hand, performing online data analysis in a centralized environment has been explored by numerous service portals such as Netflix[4]. However, the same approaches do not work in enterprise data centers because of several reasons. First, businesses such as Netflix, Facebook, and Google have the means and the scale to perform data analytics on ten thousand of production servers. More importantly, all these production servers are of similar hardware and configuration, and therefore once a technique is identified, it can be used at scale. In contrast,

most enterprise deployments and private data centers are small silos of server deployments. Hardware configurations vary significantly across enterprises. Therefore, there cannot be a one algorithm or one solution that can work in all cases. Hence troubleshooting performance issues at an enterprise deployment becomes an exercise for a domain expert to use a variety of performance monitoring tools to arrive at a logical conclusion.

However, in recent years, the pervasiveness of the cloud, and connectivity of systems to it, has opened many opportunities to improve the quality of support that enterprise companies offer to their customers. Many enterprise companies have taken advantage of such frameworks to provide faster resolution of bugs such as performance issues to customers. Prime examples of such efforts are Nimble Storage, which has developed its product called Infosight Predictive Analytics [2] and Nutanix[3] which delivers infrastructure analytics as part of its PULSE product.

Our approach shares the same goals as Nimble and Nutanix, but with the enhanced goal of being a fully cloud based solution. We believe our solution is novel because it defines an approach to send data from heterogenous end systems deployed at vastly different datacenters to a centralized cloud where the data can be processed in real time. Being completely cloud delivered buys us three significant advantages:

(i) Big Data: We can correlate datasets coming in from a large number of diverse product deployments at a centralized point and thereby provide richer analysis.

(ii) Rapid Improvements: We can provide a rapid churn of improvements in terms of the logic behind performance analysis, independent of VMware's product release cycle. As an example, once we have confidence in our ability to diagnose a new category of performance issue, we just need to update our code and catalog of issues, both reside in the cloud.

(iii) Centralized monitoring: We provide an approach to monitor how different rules perform on different heterogenous systems. This ability gives us the game changing ability to design new analytic solutions.

1.2 Our contributions

This paper is organized as follows. In Section 2, we describe the overall architecture of the Cloud Performance Analytics Framework. In Section 3, we deep dive into individual components. Section 4 showcases how this framework was used to build vSAN performance diagnostics, which is a shipped performance debugging platform for vSAN. We specifically discuss how having performance data made available in a central place enabled us to write rules for triaging common performance issues. In Section 5, we describe some of the challenges we faced in designing this framework. Section 6 concludes the paper.

2 Architecture

Figure 1 describes how a sample performance analyser tool (Perf*Analyzer) (drawn with blue box) works using the Cloud Performance Analytics framework. The diagram is color coded as follows. Two cloud services color coded in brown are used by our solution: (i) A Key-Value datastore to store results of a transaction (An example of which is Amazon DynamoDB[10]) and (ii) A SQL database to store detailed data and results (An example of which is Amazon RDS[11]) for PostGreSQL. A monitoring solution such as VMware Wavefront[12] can be used to visualize how the Perf*Analyzer tools work in run time.

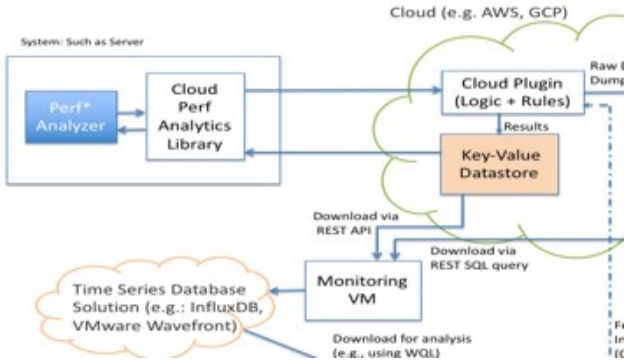

Figure 1: Overall architecture

Let us first outline the high-level steps a user must follow to use this framework.

Develop the Perf*Analyzer tool which provides the user interface, collects the necessary performance data, and interacts with Cloud Performance Analytics library.

Write rules which can process the performance data. The user may write simple rules which check for threshold violations. The user can then combine multiple violations and define when a performance exception is triggered.

If necessary, write a custom exception analyzer to implement specific semantics in analyzing performance data which might be required for this tool. The custom exception analyzer is required when simple rules are not sufficient to describe performance exceptions, and a more detailed analysis is required. The plugin allows the user to write this analysis in a programmable way in python syntax, while defining the libraries to support defining the semantics. Refer CustomExceptionAnalyzer in Section 3.4 under ExceptionRule.

If necessary, add a pre-trained Machine Learning (ML) Model or linear/ logistic regression-based rule. Our library allows an easy way to load pretrained ML models such as neural networks and run inferences on performance data to generate performance issues, which may not have been easily captured by threshold based rules or programmatic rules.

If necessary, fine tune rules, such as thresholds, after monitoring the performance exceptions generated on a monitoring solution such as Wavefront.

Our goal of building this platform is that a developer should be easily able to write, deploy, and release his own performance analyzer tool, with the help of the framework that the library provides.

3 Detailed Description

3.1 Workflow

The user endpoint of this framework is at the perf analyzer. The user invokes the perf analyzer with some configurable input parameters such as the duration of analysis desired. The perf

analyzer then collects the performance data for the desired duration and feeds it to the cloud perf analytics library. The cloud perf analytics library first marshals the data into a common JSON format and sends it across to the Cloud along with a transactionId, and then waits for the analysis. The Cloud Plugin contains the meat of the analyzer logic, and processes the data based on some defined rules or ML models. Once the analysis is complete, the result is stored in the Key-Value datastore for fast retrieval, while the payload is stored in a SQL database (DB). The cloud perf analytics library now recovers the result using the transactionId. It then un-marshals the result into a defined syntax that the perf analyzer can understand. The perf analyzer then does the final job of presenting the result to the user with a great user interface.

A monitoring Virtual Machine (VM) picks up data from both the Key-Value datastore and the DB, and then streams this data to a monitoring solution. The data can now be visualized in a dashboard by an administrator, who can infer if the analytics rules are performing as desired. The data is invaluable to improve the quality of data analysis.

3.2 Cloud Perf Analytics Library

This library provides set of APIs that the perf analyzer can use to communicate to the Cloud Plugin which contains the crux of the code for the performance data analysis. As part of its workflow, the APIs provides interfaces to: (i) Take raw performance data as an input, (ii) Convert it into a common format as expected by the Cloud Plugin, and (iii) Send the data to the Cloud, wait for the analyzed results to be available and then provide the same to its calling entity. A diagram of how this library is designed is in Figure 2.

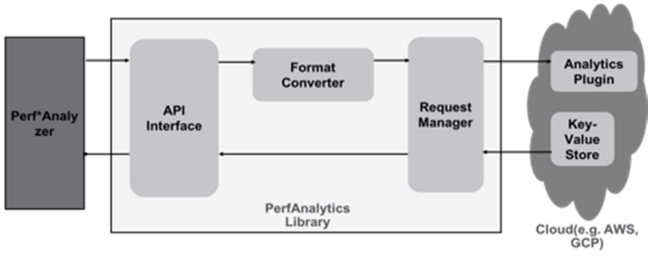

Figure 2: Components of the Cloud Perf Analytics Library

3.2.1 API Interface

The Cloud Perf Analytics Library provides a simple set of python APIs which can be used by any module/tool (or Perf*Analyzer) by simply importing library and calling the appropriate method/API as per the requirement. In response to the call, the API returns the exceptions (if there is any abnormality in the data as per the evaluated rules). Each exception will have the exception name, and the component name with associated data set (which resulted in violation of rules). The same can be consumed by the Perf*Analyzer.

Here are the definitions for the APIs provided:

Add performance data along with a data convertor

def addPerformanceData(self, context, data, dataConvertor)

@param context: Here, context in general defines the set of information which is important for analysis, besides the input data. It is in the form of a dictionary containing background information that may be necessary for performance analysis, such as, the type of system(s), the number of machines in the cluster, and clock frequency of machines stored as key value pairs.

@param data: This is the input performance data which needs to be analyzed.

@param dataConvertor: Routine to convert the performance data into standard dataformat that cloud plugin can digest.

Invoke Cloud Plugin to analyze performance

def analyzePerformance(self, isTest=True)

After the input performance data gets ready as per the required format and input context gets embedded into that, then data is sent to cloud environment for analysis.

@param isTest If set to True, invokes a test environment instead of production environment

@return dictionary with keys as performance issues and values of supporting data

3.2.2 Format Convertor

As discussed in the above (API Interface) section, the input performance data needs to be converted into a predefined JSON format on the basis of which generic rules can be written, along with context embedded into that data.

Hence, this routine is responsible for converting the input data and context into the required standard JSON format that the Cloud Analytics Plugin can understand. At a high level, the standard JSON template looks like the following:

```
{
   <Header containing details about type of performance analysis and context as Key Value entires. e.g.>
   "Plugin-Type": "performance-analysis"
   "TransactionId": "<uuid>"
   "Query Start Timestamp": <ANSI time format>
   "Query End Timestamp": <ANSI time format>
}
{
   <Time Series data for an entity for which performance data is available. e.g.,>
   "entityId": "pcpu-xx-utilization"
   "label": "Used %"
   "data: {
        "timestamps": [<Times in ANSI time format>]
        "values": [<List of floating point data>]
   }
}
```

3.2.3 Request Manager

The Request Manager is responsible for sending the request to Cloud Analytics Plugin for analysis and retrieving the analyzed response for the same. After sending the request, it validates the sending response. It then waits for the response (for configurable timeout duration) from Cloud Plugin by polling the key-value store using the transactionId (which was added while sending the request) and then returns the response to the calling entity.

3.3 Cloud Plugin

The role of the Cloud Plugin is to process the data that is sent by the Cloud Perf Analytics library and evaluate if there are any performance issues. At a high level, the plugin evaluates the data for multiple rules, which are written by the user. We have written a performance diagnostics library that can process time-series data.

3.4 Performance Diagnostics Library

The Performance Diagnostics Library (PDL) is a framework that allows a user to write rules to evaluate and analyze time-series data, and then programmatically define how performance issues may be raised when a combination of rules is violated. Our goal is to release this library as open source in conjunction with publication of this paper. PDL allows a user to construct rules in the following way.

Violation Rule

A violation rule is a simple way to express if a specific data set violates a specific threshold. We express a violation rule as a dictionary in python, which is saved in JSON format. An example of a violation rule is as follows:

```
{
   'name': 'cache-hit-check',
   'entity': 'disk',
   'metric': 'cacheHitRate',
   'threshold': 80,
   'thresholdType': 'lower',
   'minTimesForThresh': 5
}
```

This rule implies that a violation is seen anytime the cacheHitRate (metric) of any disk (entity) is lower (thresholdType) than 80% (threshold) at least for 5 (minTimesForThresh) consecutive intervals. When this rule is processed against a dataset, it will flag all those disk (s) (entities) for which the cache hit rate falls below 80% at any instance of time. This is an example of a very simple rule; in addition, the PDL offers the following constructs to evaluate data in more complex ways.

Statistical operators on metrics: Instead of evaluating individual data points for a metric, one can express an operator on all entities for a metric. As an example, find out the (moving) average read latency, (moving) 95 percent latency, standard deviation of latency, etc., across all disks, and compare with thresholds defined for these data points.

Aggregator operator on different metrics: This operator allows one to operate on two different metrics to create a new one. As an example, one can divide throughput with IOs per second (IOPS) to construct IO size, and compare that against a threshold.

Aggregator operator on different entities. This operator is very useful for distributed systems such as VMware Virtual SAN (vSAN) which have several layers of software/hardware from which metrics and data points are derived. As an example, one can divide the read latency seen at the vSAN host-domclient layer (software layer which interacts with the Virtual SCSI (vSCSI) layer with the read latency seen at the disk layer. This result gives the latency increase (inflation) in the vSAN stack. A similar operand can be used to determine IO inflation in the vSAN stack. These calculated data points can now be compared against a threshold in the same manner as any other violation rule.

Exception Rule

An exception rule defines the conditions during which a performance exception is triggered. The exception rule is triggered when one or multiple violation rules flag entities that have violated performance expectations. An exception rule can be defined in two different ways.

i. Mapped Exceptions: For simplicity, we defined a 1-1 mapping of an exception rule with a violation rule, that is the violation directly triggers an exception. We maintain this in a RuleToExMapper file. So, writing a new rule and mapping it to corresponding new exception is simple. Hence extending the rule-set does not require any in-depth knowledge of framework for any user.

ii. Custom Exception Analyzer: This allows one to combine multiple violations (with different semantics) and support implementation of flowcharts using them to trigger exceptions. The custom exception analyzer framework provides a lot more flexibility in terms of combining various rules as per the runtime dynamics and implementing a complete flow chart using this support.

PDL provides an ExceptionAnalyzer class which defines all the standard APIs that any custom exception analyzer would need. APIs include (i) Methods to which Violation Rules are not met (with their names), (ii) Methods to apply logical conditions on violated rules (e.g., performing logical AND on all rules in list. A new custom analyzer can easily be added by extending ExceptionAnalyzer, and this custom analyzer gets automatically registered with the Cloud Analytics Plugin and invoked when data of the corresponding performance analysis type is received.

Dynamic thresholds and regression:

A key limitation of the violation rule is that a threshold needs to be defined to trigger a violation. In some cases, such as a cache hit rate or CPU utilization, thresholds can be intuitive based on systems knowledge and experience. In some other cases, a threshold can be derived by gathering a lot of data points and then applying clustering algorithms, such as the one described in Section 4.1.3. However, in many cases, it is near impossible to define a threshold value that uniformly applies to all kinds of systems, hardware, and software that a generic performance analyzer needs to work on. One of the main advantages of a cloud framework is that performance data is centralized at one place, and therefore readily available for many kinds of analysis including building data driven. PDL offers two different techniques to eliminate thresholds.

1) **Dynamic thresholds**: With dynamic thresholds, a system learns the threshold by analyzing data over a period of time, and then uses this prediction to trigger violations. In general, dynamic threshold techniques [9] need a large amount of data along with a human input to guide the approach in the correct

direction. PDL supports this technique where the threshold can be defined as dynamic, instead of assigning a value to it.

2) **Linear Regression Analysis**: In this technique, PDL identifies outliers of a certain metric by building a regression model of predicting the metric's value as a function of the values of certain other metrics and constructing a band of acceptable ranges depending on the prediction and standard error of forecast of regression model. When the actual data point for the predicted metric falls outside the band, a violation is triggered. Our framework encapsulates a regression model in a standard INI syntax.

3.5 Monitoring

A big challenge in an analytics-based approach is to understand if the rules are correctly identifying valid performance issues in a production deployment. We leverage the power of the cloud to provide a platform where it is possible to centrally visualize how rules are functioning for production deployed systems in real time. Our goal is to build a centralized data lake where we can store all datasets generated by Perf* Analyzer tools, and corresponding results so that we can continuously monitor and improve rules. Please refer to Figure 1 for the design. A VM polls the input data and corresponding results from the Key-Value store and PostgreSQL database, processes them to collect vital statistics, and then posts these statistics along with the raw dataset to a commercial time-series database, such as a Wavefront instance. For each performance issue generated by the Perf* Analyzer tool, we generate the following statistics:
(i) The number and type of performance exceptions.
(ii) Specific data points of the input dataset such as aggregating the IOPS, throughput, and latency.
(iii) Raw input dataset.

(i) and (ii) can be stored as entries in the time-series database. (iii) can be stored as a JSON blob attached to the entry.
The data can be used in two ways. Wavefront provides us with a way to visualize all the time series data in a dashboard with a browser. Section 4.1.5 highlights on how the Wavefront dashboard helps in improving the rules. Wavefront also has an option to query the data using the Wavefront Query Language [8]. In sections 4.1.3 and 4.1.4, we explore how the data can be used in building some Machine Language (ML) models.

4 Applications

4.1 Virtual SAN (vSAN) Performance Diagnostics

VMware Virtual SAN (vSAN) powers an industry-leading Hyper-Converged Infrastructure (HCI) solution with a vSphere-native, high-performance distributed storage architecture. As with any distributed system, performance bottlenecks may occur at multiple choke points. vSAN Performance Diagnostics provides feedback on how to avoid these choke points and thereby how to extract the best performance in a given vSAN cluster. vSAN Performance Diagnostics is available as a feature in VMware vCenter Appliance (which is VMware's control and management framework used for managing VMware products such as vSAN). It consumes the performance data available via the vSAN performance service in vSAN Health Service. The data is then sent to VMware's internal cloud where a Performance Diagnostics Cloud Plugin churns through the data and identifies performance issues. The results are sent back to vCenter and are then displayed in the performance diagnostics UI. The UI provides details on performance issues in the VSAN cluster accompanied with data, analysis, and links to KB articles to allow users to troubleshoot performance problems on their vSAN system. An illustration of how this feature works is in Figure 3. vSAN Performance Diagnostics is a feature available in VMware vSAN since vSAN 6.6.1.

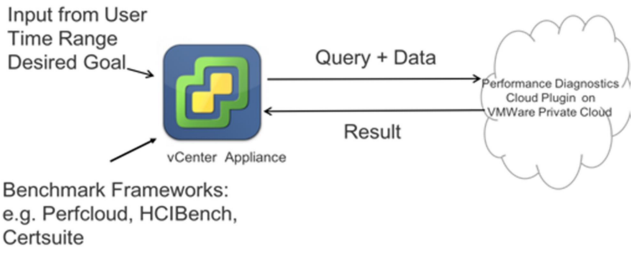

Figure 3: Illustration of vSAN Performance Diagnostics.

In this section, we will explore how we wrote different kinds of rules on the cloud plugin that processes vSAN performance data and briefly explain the utility of monitoring dashboard.

4.1.1. Benchmark oriented feedback

It is very typical of a vSAN customer to run a standard open source or commercial storage benchmark such as FIO[6] or HCIBench [5] to evaluate the performance of the system. However, this is often a stumbling block, because naïve users usually lack the competence to get the best performance of a system from a performance benchmark. As a result, users may associate an underperforming benchmark with a poor impression of the system.
We have written benchmark specific rules to identify changes that may be made in a benchmark to get a better performance from the system. First, we supply the benchmark workload in the context field of addPerformanceData(). Next, we write rules specific to our understanding of how the benchmark should optimally perform. Some examples include:
(i) Tuning the number of VMs to get a better result (Larger number of VMs results in higher throughput, smaller number of VMs results in lower latency), (ii) Tuning outstanding IOs issued by the benchmark (where current value is low). In all such rules, threshold is computed based on an understanding of how a benchmark is expected to perform best. As an example, in the case of HCIBench running on vSAN, we need at least one VM disk placed on every physical disk for best performance, else the physical disk remains unused. Therefore, this rule simply checks if there is disk activity on every physical disk.

4.1.2 System issues identified using domain knowledge:

These rules are derived based on a system expert's knowledge of how a vSAN system is expected to perform. As an example, vSAN relies on a variety kernel threads to do IO processing. If the CPU utilization of any thread goes to close to 100%, it will

imply that the thread is running out of CPU, and in that case the latency of an IO transaction would be higher than expected. Similarly, if the thread ready time (The time that a thread is waiting in queue before the scheduler can run it on a physical CPU) crosses 5%, it indicates starvation, which can lead to higher I/O latency. In such cases, it is easier to write a rule, because the meaning of a threshold is clear.

4.1.3 Deriving thresholds for complex issues

For many performance issues, it is non-trivial to write a simple rule such as those in 4.1.1 and 4.1.2. In the case of vSAN, an example of a complex issue is the network dropping packets or a disk having checksum errors. Determining a threshold for these kinds of issues is challenging. As an example, a network can usually tolerate some degree of errors (because of TCP's functionality), without affecting the application/ system performance. We desire to determine a threshold for the packet drop rate beyond which there is a high impact to system/ application performance.

In such cases, the cloud based approach becomes handy. Since, we collect data in a central data repository (VMware Wavefront database in our case) for every invocation of vSAN performance diagnostics, over time, this data source becomes very rich and represents samples of thousands of vSAN production clusters running in our customers datacenters. We use this data offline for building/training ML models and then add them to PerfAnalytics Plugin. For example, using this data, we can learn how the write latency of a vSAN IO transaction changes as a function of the TCP retransmission rate.

We separately analyze for the two different flavors of vSAN clusters: (i) The hybrid clusters where the disks are traditional Hard Disk Drives (HDDs), and (ii) The All-flash clusters where the disks are flash/ Solid State Drives (SSDs). The cumulative sizes of the dataset exceed 1 Million data points from over 1000 deployments and are shown in Table 1.

Type of vSAN Cluster	Number of unique systems	Number of datapoints
Hybrid vSAN	933	1184836
All Flash vSAN	215	354966

Table 1: Characteristics of our data set

We first normalize the data using the Z-score normalization technique. We then apply the K-means clustering to the resulting dataset. Figure 4 and Figure 5 show the result of the analysis for the two flavors of VSAN clusters. The clusters identified by the K-means clustering algorithm are color coded as described in Table 2.

The clusters identified by yellow and green regions represent the scenario we want to detect: where the TCP retransmission rate actually affects vSAN I/O latency. We identified the lower TCP transmission threshold as that marked by the green circles. The values for the two types of clusters are in Table 3. We use these values as thresholds in our rules to detect when high TCP retransmission rates could be affecting system performance.

Color	Representation
Black	Normal region of operation
Orange	High Latency Uncorrelated with TCP Retransmit Rate
Red	Extreme case of issues in Orange cluster
Yellow	High Latency Correlated with TCP Retransmission Rate
Green	Extreme case of issues in Yellow cluster

Table 2: Definition of color representations after clustering

4.1.4 Regression to find latency outliers.

While the ideas in 4.1.1, 4.1.2, and 4.1.3 are good, they rely on human systems knowledge and expertise for design of the rules. This has a fundamental problem, often systems behave in very different ways than humans imagine, therefore many performance issues that happen in the field go unnoticed and undiagnosed. In this section, we explore how a simple linear regression can be built to indicate issues where there is significant increase in latency in the vSAN stack.

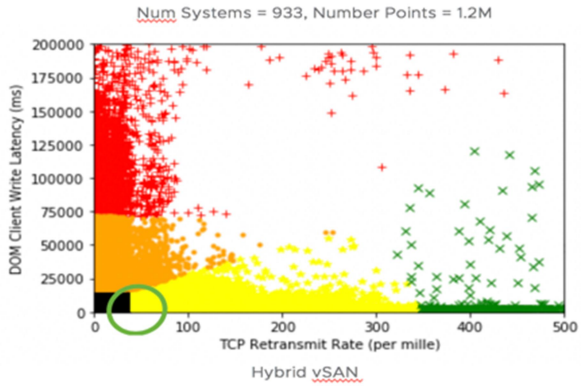

Figure 4: K-means clustering result for Hybrid vSAN

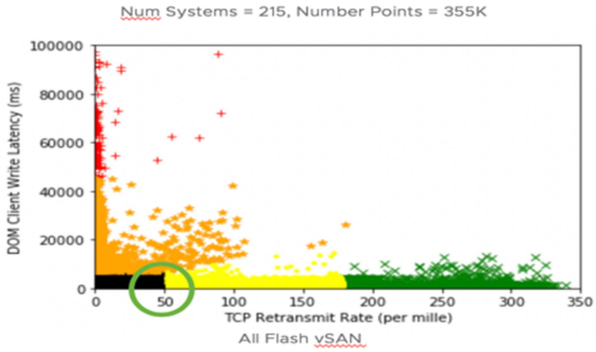

Figure 5: K-means clustering result for All-flash vSAN

Type of vSAN Cluster	Threshold value determined from K-means clustering
Hybrid vSAN	38
All-Flash vSAN	49

Table 3: Results from K-means Clustering

Fundamentally, if we consider a vSAN system as a blackbox, we can consider that latency for an I/O transaction can be described as a function of few metrics that describe the state of the system. These metrics are: (i) IOs per second (IOPS), (ii) Throughput, (iii) Outstanding IOs, and (iv) Congestion (A signal used by vSAN to throttle in incoming IO rate). In this approach, we build a regression model to predict the behavior of write latency as a function of the above variables. Concretely, this approach can be summarized as follows.

(1) We first separate the data points into all-flash and hybrid vSAN clusters, and normalize the data using K-score normalization. This is identical to the technique used in Section 4.1.3. Similar to Section 4.1.3, we focus our efforts on write IO, therefore the IOPS, Throughput, and OIO metrics captured are specific to write IOs.

(2) **Regression Equation:** We define the regression equation as follows:

iii. $$Lat = B + W_{IOPS}IOPS + W_{Tput}Tput + W_{Cong}Cong + W_{OIO}OIO + W_{\sqrt{Cong}}\sqrt{Cong} + W_{\sqrt{OIO}}\sqrt{OIO}$$

Where B is a bias and W's are the weights for the respective variables. We solve the linear regression for minimizing the mean square error with a higher order regularization term. We solve this equation independently for the two classes of vSAN clusters with the Adam Optimizer available in the Tensorflow library.

(3) **Standard Error of Forecast:** After the regression model is solved, we derive the standard error of forecast using standard statistical techniques.

(4) **Predicting latency outliers**: Once the regression is solved, we employ it as follows. For any test dataset, we normalize the data using the same values for the K-score normalization defined in Step (1). Then, we calculate the predicted value of write latency using the solved equation in Step (2). We then measure the actual value of write latency relative to the predicted value. If the actual value is outside the standard error of forecast, then the data point is considered an outlier and an issue is flagged.

We evaluate the accuracy of the regression model as follows. We divide the 355 K data points for All-Flash clusters described in Table 1 into three buckets:

iv. Bucket 1: Write Latency lower than 10ms. Such points are definitely not abnormal as vSAN write latency is expected to be in this range.

v. Bucket 2: Write Latency greater than 100ms. Such points are definitely outliers.

vi. Bucket 3. All other points with IOPS > 100 (To avoid idle clusters). These points are hardest to classify.

We first run the regression model on points in Buckets 1 and 2. Since these points are already classified, it is easy to compare whether regression yields a correct result. The rate of false positives (defined as the percentage of normal data points being tagged as anomalies) and false negatives (defined as percentage of anomaly points which were missed by the algorithm) is described in Table 4.

	Bucket 1 + Bucket 2
False Positives	0.68 %
False Negatives	21.96 %

Table 4: Rate of False Positives and False Negatives from Regression Model for All-Flash vSAN clusters.

We then pick 200 random points in bucket 3 and manually classify them as normal (126 points) or abnormal (74 points). This is a good mix of normal and anomalous points, and therefore serves as a healthy sample for evaluation of the regression model as a classification technique. After applying the regression model on these 200 points, 150 were classified as normal, while 50 were classified as anomalies. Out of these 50 data points, 42 were true anomalies, while 8 were false positives. Using these numbers, Table 5 shows the calculated precision, recall, and F-score of the regression model.

Score	**Bucket 3**
Precision	84%
Recall	57%
F-Score	68%

Table 5: Precision, Recall, and F-Scores for Regression

We were impressed by the low rate of false positives for Bucket 1 +. Bucket 2. Similarly, in Bucket 3 where the hardest challenge lies, the precision stands at 84%. This gives us confidence that when an issue is raised for anomalous behavior, it is indeed an issue. At the same time the rate of false negatives for Bucket 1 + Bucket 2, and the lower recall score for Bucket 3 implies that there is room to improve, since there are many performance issues that go undetected.

The regression model also gave us new intuitions on the way vSAN systems behave. As an example, for hybrid vSAN, the dominant weight turns out to be congestion. On the other hand, for an all-flash vSAN, the dominant weight is Outstanding IOs. This can be explained from a systems point of view as follows. vSAN is a two-tiered storage architecture, where writes IOs are first written to a cache tier. Thereafter, a background thread known as the elevator, moves the data from the cache tier to the capacity tier and makes it persistent over there. This process is formally defined as de-staging. In a hybrid vSAN, the rate of de-staging is much lower (because the capacity disks are spinning magnetic disks). Therefore, the vSAN system raises congestion when subjected to heavy load, to throttle the incoming IO Rate. This throttling mechanism leads to latency buildup due to queuing at the upper layers. Hence congestion is the dominant contributor to latency in a hybrid vSAN. However, in an all-flash vSAN, the de-staging rate is much faster owing to SSDs in the capacity tier. Therefore, congestion is hardly raised, instead the dominant factor to latency becomes the number of outstanding requests, which raise the possibility of longer wait times on queues before the IO is formally processed.

Deploying vSAN Performance Diagnostics also gave us insights to hidden bugs in our software. VMware products such as vSAN are deployed on a very large range of hardware and mix of compute, memory, and storage specifications. Hence, there is a potential of software bugs causing performance issues in some systems, on which the software has not been internally tested

yet. Over a period of a year, we identified five such bugs, and have been able to quickly fix them in the next version of our product.

4.1.5 Monitoring with Wavefront

In Section 3.5, we described how the input data and results are posted to Wavefront. Figure 6 shows a small snapshot of the Wavefront dashboard. Each dot on the dashboard shows a specific performance issue generated, color code by the uuid of the environment it is generated on. The Y-axis shows the number of instances of performance issues. This dashboard can be useful to an administrator in many ways. As an example, if a specific exception is occurring in almost all deployments in quick successions, the rule may be yielding high false negatives, and we may need to modify such rules with different thresholds. It also helps us in understanding importance of a rule, by using some simple analysis we can plot a histogram of the distribution of exceptions under different feature categories. For example, we detected that benchmarking specific rules are frequently triggered, which led us to spend more energy on some of the benchmark specific rules.

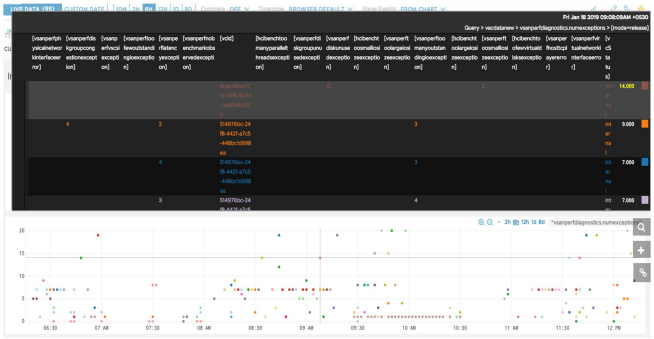

Figure 6: Wavefront Dashboard View

5 Challenges

While sending performance and systems data to the cloud brings a lot of advantages, it also exposes its own challenges. We document some of these challenges in this section:

(1) **Connectivity to the Cloud**: In many cases, data centers are behind a firewall with no connectivity to the Internet, mainly for security reasons. In such a case, the Cloud Request Manager will not work. The solution is to deploy a private cloud within the customer's private data center where the Cloud Plugin can work; and ship the rules to the Cloud Plugin in an offline mode. This is a work in progress for us and is not available yet.

(2) **Privacy:** In many cases, a customer is not comfortable to send systems data to a cloud because of privacy issues. As an example, in certain cases, the names of systems may be needed to be confidential. This is not a big problem for our framework, because we need time series data for different entities and metrics. Anything else can be obfuscated.

In VMware, we have an agreement known as CEIP [8], which defines what information VMware can collect from its customers and what not. It is an opt-in agreement, customers are encouraged to opt-in, but this is not enforced. vSAN Performance Diagnostics respects CEIP; however, if CEIP is disabled, the vSAN Performance Diagnostics would simply not work.

(3) **Cost:** While the volume of zipped performance data is small (of the order for a few Mbytes), we recognize that in certain cases (as an example when the time duration requested for performance analysis is very long), the volume of the data can be of the order of 100's of MBytes. In such a case, there is a cost to send this volume of data over the network. As an example, the volume of traffic may hinder another critical network traffic. As per our opinion, this is a necessary and reasonable cost to pay for such tools and should not be an impediment.

6 Conclusion

In this paper, we document a framework of supporting cloud-based performance analysis and debugging tools. We demonstrate how these tools can be developed and show some interesting use cases where they have shown promise.

REFERENCES

[1] VSAN Performance Diagnostics Knowledge Base at https://kb.vmware.com/s/article/2148770

[2] Nimble Storage Infosight Predictive Analytics: https://www.adn.de/fileadmin/user_upload/Hersteller/Nimble/Datenblaetter/nimblestorage-ds-infosight.pdf

[3] Nutanix: Improving customer experience with Analytics: https://www.nutanix.com/2017/04/20/improving-nutanix-customer-experience-analytics/

[4] "Tracking down the Villains: Outlier Detection at Netflix", at https://medium.com/netflix-techblog/tracking-down-the-villains-outlier-detection-at-netflix-40360b31732

[5] HCIBench at https://labs.vmware.com/flings/hcibench

[6] Flexible I/O Tester by Jen Axboe at https://github.com/axboe/fio

[7] Wang, C., Kavulya, S.P., Tan, J., Hu, L., Kutare, M., Kasick, M., Schwan, K., Narasimhan, P. and Gandhi, R., " Performance troubleshooting in data centers: an annotated bibliography?" ACM SIGOPS Operating Systems Review, 47(3), pp.50-62.

[8] Wavefront Query Language (WQL) at https://docs.wavefront.com/query_language_reference.html

[9] Marvasti, M.A., Poghosyan, A.V., Harutyunyan, A.N. and Grigoryan, N., 2014. An Enterprise Dynamic Thresholding System. In ICAC (pp. 129-135)

[10] Sivasubramanian, S., 2012, May. Amazon dynamoDB: a seamlessly scalable non-relational database service. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (pp. 729-730). ACM.

[11] Amazon Relational Database Service (RDS) at https://aws.amazon.com/rds/