MAVIS: Managing Datacenters using Smartphones

Raghavendran Shankar,* Benjamin Kobin,* Matthew Tancreti,* Saurabh Bagchi,* Michael Kistler,† Jan S Rellermeyer,†
*Purdue University, {rshankar, bkobin, mtancret, sbagchi}@purdue.edu
†IBM Research Austin, {mkistler, rellermeyer}@us.ibm.com

***Abstract*—Distributed monitoring plays a crucial role in managing the activities of cloud-based datacenters. System administrators have long relied on monitoring systems such as Nagios and Ganglia to obtain status alerts regarding the state of computing infrastructure on their desktop-class machines. However, the popularity of mobile devices is pushing the community to develop datacenter monitoring solutions for smartphone-class devices. Our work tackles the problem of a monitoring system geared toward mobile devices, which we call MAVIS. We show that existing desktop-oriented solutions are either hugely inaccurate or drain the resources (bandwidth and compute) of mobile devices. MAVIS introduces novel complex event processing, partitioning between the monitored hosts and an intermediate server, and pinpointed notification to mobile devices. We do an evaluation with real metric data accumulated over 1 month from a university datacenter.**

I. INTRODUCTION

Cloud computing has become a near ubiquitous model for delivering computing services. In it, application developers can host their applications and services on third party infrastructure. This model has attracted a lot of businesses and developers to host their applications and services in cloud-based datacenters. Some of the popular choices for hosting applications include Amazon Elastic Cloud Compute (EC2), Microsoft Azure, and Google Cloud Platform. However, maintaining and managing cloud datacenters is a critical and challenging task for these cloud vendors. If a cloud vendor does not provide a high degree of availability for the machines in its datacenters, the applications hosted on them will have outages, and this could lead to loss of reputation and revenue for the cloud vendor. As a result, systems management of commercial datacenters is a critical task and one where improvements are being rapidly made and rolled out.

In order to maintain and manage datacenters, administrators monitor a variety of performance metrics from various levels of the software stack, such as utilization of CPU, memory, and network resources, to (higher in the stack) the frequency of garbage collection in a Java or C# runtime. There exist tools to aggregate raw data and create higher-level information so that the admins can take prompt action. Nagios [6] and Ganglia [7] are two of the most popular commercial systems which are utilized today for systems management. They can provide raw metric values and also compare values against thresholds to generate alerts for simple subscriptions. They also provide graphical outputs for human consumption.

While systems management through desktop interfaces is the norm today, increasingly the need is being felt for performing systems management through mobile devices [4], [1], [5]. This need is arising increasingly often due to three reasons — the first is the ubiquity of smart phones, the second (caused in part by the first) is the mobile workforce of today, and the third is the need for responding promptly to outages or impending outages in the cloud infrastructure. The second factor speaks to the fact that the workforce is no longer tethered to desktop-class machines for all of their workdays.

Let us first consider the challenges in monitoring datacenter activities. In general the primary challenges are:
(i) The monitoring data must be processed quickly and anomalous or suspect events must be detected quickly.
(ii) The overhead of monitoring must be kept to a bare minimum on the machines in the datacenter, so that they can perform their main function as the workhorses for running the workloads.

In addition to these, performing systems management through mobile devices has some specialized, and equally crucial, challenges:
(i) The mobile devices are resource constrained, significantly in battery and bandwidth. They cannot be relied on to gather huge amounts of data and perform sophisticated data analysis on the device.
(ii) The tools available on mobile devices are quite limited and some of the available ones do not match their user interface to the constraints of mobile devices. For example, some tools create extensive graphs which may be difficult to discern and interpret on mobile devices.

Solution Approach: *We argue that a three-pronged approach addresses the above-mentioned problems of systems management from mobile devices. First*, an intermediate server placed between the target machines (the datacenter machines) and the mobile devices can perform much of the functionality of parsing monitored data and creating actionable information out of them. Related to this is the necessity for intelligently partitioning the systems management rules among the various target machines and the intermediate server. *Second*, we need a rich subscription language specialized for expressing events of interest for systems management, *e.g.*, patterns in metric values, either individually or together with another metric. By doing a survey of system administrators at a tier-1 research university and an industrial research lab, we have discerned that the current simple subscriptions are inefficient to express many common systems administration tasks. *Third*, we need efficient processing and communication primitives respectively for processing the event streams at the intermediate server and communicating information with the mobile devices.

We achieve all of these design goals in a system that we call MAVIS, which is a system that can support managing large scale datacenters from mobile devices. The introduction of an intermediate server enables us to design and support a rich subscription language specialized for systems management, which would not have been possible on the mobile devices. Thus, there is a natural, albeit subtle, interplay between the two thrusts of our work — mobile management and semantically rich subscription processing system. In order to create such a system, we utilize a streaming event processing engine from IBM called *Infosphere Streams*, or simply *Streams*, for running subscription queries against streaming monitoring data. While streaming event processing engines have been used to support simple subscriptions for desktop monitoring [20], they prove to be even more valuable to support the rich subscription language which we present here for mobile systems management. We discuss this importance further in VIII-A

In the design and implementation of MAVIS, we introduce three novel components that interact with the streaming event processing engine — selective storage, selective notifications, and selective offloading. Selective storage refers to selectively storing metric values to provide context information for problem analysis when there is an alert in the system. Selective notifications enable us to efficiently disseminate data to the mobile end points without consuming excessive bandwidth. Selective offloading enables us to improve the scalability of our event processing engine, by offloading some processing to the datacenter nodes. At the intermediate server, we have a two-layered architecture for event detection which helps to make MAVIS a scalable architecture for monitoring medium-sized datacenter activities from mobile devices.

This paper makes three novel contributions.

1) MAVIS intelligently partitions the work among the actual datacenter nodes and a complex event processing(CEP) engine. It uses a highly efficient streaming event processing sub-system in the CEP engine which lets it handle a large number of subscriptions, while it maintains the monitoring and systems management overhead at the target nodes to a minimal level.
2) MAVIS shows how systems management of a large number of datacenter machines can be done from mobile devices. Toward this goal, it shows how to limit the resource utilization (chiefly bandwidth and compute) at the mobile devices while providing actionable information to the mobile device user.
3) MAVIS supports the richest domain specific language to date for subscriptions for triggering alerts for systems management tasks. Our language design is driven by a survey of systems management tasks conducted among system administration professionals.

For evaluation, we compare MAVIS to the current state-of-the-art mobile monitoring solution, the highest rated application from Google Play Store, called *aNag*, which is billed as a Nagios client for Android devices. We also compare MAVIS to a research prototype named Volley [15], which can provide efficient monitoring of invariants. Our evaluation brings out a few important insights, some of them unexpected. First, MAVIS can deliver alerts to a mobile device in less than 10 seconds end-to-end with 231K subscriptions in the system. Second, Volley's strategy of decreasing sampling frequency misses an unexpectedly large percentage (almost 40%) of events of interest that happen to be rare and short-lived, even when the user-specified maximum tolerable error is 5%. Third, the aNag solution consumes high bandwidth (2 MB/minute) while monitoring just 130 target machines. Finally, MAVIS's technique of creating distributed CEP engines enables it to scale up. CPU utilization becomes the driving factor to force distribution and a single instance of the MAVIS CEP engine running on one machine can handle 231K subscriptions.

II. PROBLEM BACKGROUND

Datacenter management involves continuous monitoring of system resources. For example, consider a scenario where an administrator would like to take control actions when there is a high CPU load on one of the nodes in a datacenter. In such a scenario, the administrator can create *subscriptions* for alerts when the CPU load on a node exceeds a threshold value. This example can further be expanded to managing a set of nodes in the datacenter that are all located in the same server rack. Hence, there are mainly two types of subscriptions – (a) *local subscriptions* confined to a single machine, *i.e.*, check if the CPU load on the node whose ID is 100 is greater than 80%, and (b) *spatial subscriptions* which span a set of machines i.e. check if the CPU load on each of the nodes in a subnet is greater than 80%. Spatial subscriptions form an AND relation and are only matched when all the nodes in the subscription meet the subscription criteria. This type of monitoring is reflected in a variety of existing solutions, such as Nagios, Ganglia, and Sensu. Further, subscriptions can be based on instantaneous snapshots of metric values, or use a time window full of metric values.

Modern datacenters typically monitor six to twelve metrics per datacenter node, and typically have ten to thirty system administrators, at large academic institutions (lower end of the range) and commercial organizations with data centers for internal operations (as opposed to having it as a line of business). Further, it is common for system administrators to create multiple subscriptions for a metric as they might have overlapping interests [18], [1]. For instance, one system administrator may be interested in the behavior of the hypervisors on a rack of machines, together with the temperature profile of the rack. A second administrator may be interested in the CPU load of the machines, along with the temperature profile, since the temperature profile may be affected by a variety of factors. A representative value for number of subscriptions can be arrived at by considering the following values. Each system administrator would create at most 2 subscriptions per metric, there are 10 metrics per node, and 30 system administrators giving 600 subscriptions for each physical machine. Also, there are VMs resident on each physical machine and for the purposes of monitoring, they are considered valid target end points. Thus, with 10 VMs per physical machine say, there would be 6,600 subscriptions, since metrics come from the 1 physical hardware/hypervisor plus 10 VMs. At ITaP, Purdue's IT organization, the number of virtual machines per host machine varies between 10 and 15. The point is that the number of subscriptions that must be supported by a middleware system tends to grow rapidly with the number of machines in a datacenter.

To disambiguate the various target end points for monitoring, we refer to the following constituents of each physical machine: *Physical hardware*, which has measurements from the OS running on the bare metal hardware and the hypervisor, and *Virtual machines*, which has the obvious meaning. These collectively form the *Target end points* (for monitoring).

A Motivating Example

On examining the metric data from Purdue's IT organization we realized that complex subscriptions could help detect alerts in some interesting situations. For instance, while monitoring the total number of processes running on a Linux machine we observed an instance where the number of processes suddenly spiked up. In order to identify these surges we can create subscriptions to notify us if there is a high degree of variance in the data and if the mean or median of the data exceeds a given threshold value. This kind of subscription will help administrators identify scenarios where a problem is very likely to occur due to the large variation in the recorded values. Another example that we saw in our university's datacenters is that the load across a set of machines behind a load balancer start to deviate. This happens because the load balancer has some complex assignment strategies, such as, based on the number of active connections, with weights that are re-adjusted at runtime. A race condition in the assignment strategy causes the load across the machines to get disbalanced. This can be detected only though a mix of spatial monitoring (across the multiple machines which are behind the load balancer) and temporal monitoring (observing the deviation in loads grow larger with time).

The datacenter monitoring demands of today mean that not all monitoring can be done on the target end points (takes away resources from running the actual workload), not all monitoring can be done on the mobile devices (too much resource usage on the resource-constrained devices), and not all monitoring can be accomplished with simple instantaneous rules. In [20], the authors show that aggregate operators can benefit systems management. Existing work has almost solely focused on creating scalable systems for supporting simple subscriptions [14], [15], [6], [7].

III. Datacenter management with Mobile Devices

The traditional solution to monitor datacenters is to have monitoring software tools installed on the monitored nodes and have administrators use desktop-class machines to view the monitored data. Many tools abound and are widely used in practice, including, Nagios [6], Ganglia [7], and Sensu [19]. However, the wide availability of smart mobile devices is poised to change the way system administration of large data centers is done. It is poised to do this by allowing system administrators to manage events from their mobile devices rather than being chained to the desktop monitors. Further, this also holds the potential for the system administrators to do remote troubleshooting and initiate mitigation actions from their mobile devices.

A. Current Approaches

There are a small but growing number of mobile applications that allow system administrators to manage physical target machines from the mobile devices. For example, as of December 18, 2014, there are 24 apps for systems management on the Google Play Store with ratings of 4 stars (out of 5) or higher. However, these are cases where there has been a rush to functionality with less thought paid to robustness, usability (handling complex queries that are typical of systems management tasks), or resource consumption on the mobile devices. As a growing number of system management professionals begin to use mobile devices to receive and handle monitoring alerts, it is imperative that network traffic costs be kept down, and that the battery life of the mobile device not be significantly degraded.

Current mobile management solutions cause high traffic between end users' mobile devices and the monitored host machines within the data center. This is fundamentally because they do not interpose any filter between the raw monitoring data generated by the target machines and the mobile devices which consume this raw data. For example, aNag, the highest rated mobile monitoring solution, which uses Nagios, consumes 1.5 MB/min for monitoring 100 target end points. It pulls the data raw data directly from the target machines [9]. There are some plugins with Nagios that can push the data out, but these still do not solve the problem of selectively filtering the data using a rich subscription language. Furthermore, current solutions have been directly integrated into architectures which were primarily built for desktop monitoring. For example, a plot with 5 lines in different colors may be discernible on a desktop but is likely not on a mobile screen. Furthermore, sending all of the metric data would consume excessive bandwidth. This direct integration hinders applications from efficiently obtaining more insights about the alert data.

Regarding usability, systems management tasks often require contextual information to perform the root cause analysis of failures. Many mobile management solutions provide only pinpointed alert information, without any context information. For example, aNag can provide an alert when a Nagios trigger condition (such as, CPU utilization reaching a threshold) is met on a remote machine. Such pinpointed alert information is very useful, but for many systems management tasks more information is needed to provide context around the alert. For example, in the above example, it would be important to know what the CPU utilization was on comparable machines serving the same customer's workload. While Nagios can provide this kind of information on desktop interfaces by maintaining system wide graphs, replicating this on mobile devices creates usability and excessive bandwidth consumption problems.

B. Requirements

To inform the design of our system monitoring application, we conducted a survey of system administrators at the central IT organization of a tier-1 research university and at a large industrial research lab. The results of the survey supported our assumption that a complex domain specific language (DSL) for MAVIS would be useful for systems management. The detailed results of the survey can be seen in Section VIII-B.

Through our detailed investigation, we outline the following as the primary requirements from a mobile systems monitoring solution
(i) The solution should enable users to receive timely alerts on a mobile device, provided they are connected to the network. If the user is not connected to the network, an asynchronous notification should be delivered when he/she does connect.
(ii) It should be scalable to at least a few thousands of end hosts for targeting reasonable-sized commercial data centers.
(iii) The solution should be parsimonious in the resource usage at the mobile devices; the most relevant resources being wireless bandwidth, computation, and resultantly, battery life.
(iv) The solution should allow administrators to create on-the-fly subscriptions without having to modify individual configuration files on target machines. Such on-the-fly subscriptions are needed because system administrators realize the need for new rules as the datacenter operates and they see new faults. Making changes to the target machines is often considered a very sensitive operation because it risks disrupting the workload running on the target machines.
(v) It should allow system administrators to obtain the context information for an alert, enabling them to diagnose the failure.

IV. MAVIS: DESIGN AND IMPLEMENTATION

The MAVIS framework meets the requirements of mobile systems monitoring application as outlined in the previous section. The MAVIS framework accomplishes the following tasks: managing subscriptions written in a rich domain-specific language, analyzing large volumes of streaming data, detecting the events that match current subscriptions, providing the notification to mobile devices in an efficient and timely manner, and retaining and storing data around detected events for the subscribers to retrieve on demand. These tasks are not orthogonal but complement each other, and must be designed in a jointly optimal manner for an efficient implementation of the system. For example, the detection of an event of interest is a trigger to mark data that needs to be retained long term.

Figure 1 presents an architectural diagram of the components in MAVIS. MAVIS comprises of the following components – MAVIS publisher, MAVIS intermediate server which is composed of CEP engine(s) and a subscription manager, and the MAVIS subscriber, which is a mobile application. There is a MAVIS publisher running on each datacenter node, and several MAVIS publishers are associated with one MAVIS CEP engine(there are multiple CEP engines for the entire system). The MAVIS mobile application creates subscriptions and these are sent to the MAVIS subscription manager. Once a subscription is created on the MAVIS subscription manager, it is handed off to the associated MAVIS CEP engine(s). The process of matching collected metrics against current subscriptions either happens at the MAVIS publisher or the MAVIS CEP engine.

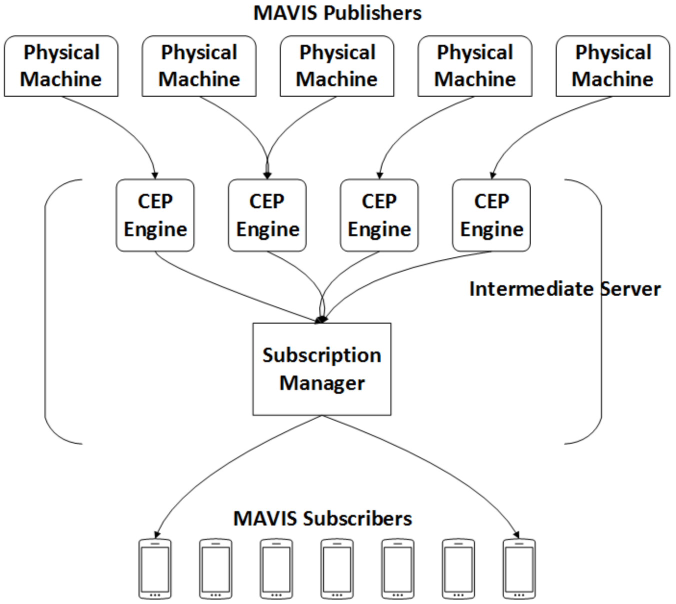
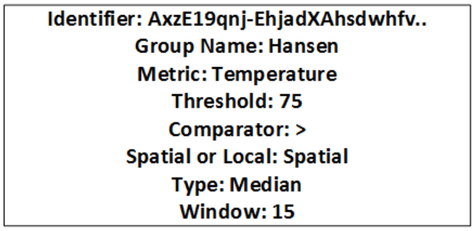


Fig. 1. Flow of events in MAVIS during event detection



Identifier: AxzE19qnj-EhjadXAhsdwhfv..
Group Name: Hansen
Metric: Temperature
Threshold: 75
Comparator: >
Spatial or Local: Spatial
Type: Median
Window: 15

Fig. 2. MAVIS subscription format

1) MAVIS *Domain Specific Language*

MAVIS offers a rich DSL which can be utilized by the MAVIS subscriber to create alert subscriptions. We categorize the subscriptions as *local* or *spatial* subscriptions. A second orthogonal dimension is *instantaneous* or *temporal* subscriptions. An instantaneous subscription is based on a single snapshot in time while a temporal subscription is based on metric values over a window of time. For temporal subscriptions, we support a variety of aggregation operators — min, max, mean, median, standard deviation, variance, percentile, etc. Some of these operators are cheap to calculate, in terms of processing and memory, such as, min and max, while others require more resources. We present the formal grammar for our DSL in the Appendix in Section VIII-D. For the subscription in Figure 2, the MAVIS subscription manager sends an alert to the subscriber when the average temperature of the past 15 monitored values on all of the host machines within the group Hansen are greater than 75 degrees. This is thus both a spatial and a temporal subscription.

2) Subscription Creation

When the mobile application creates a subscription, the subscription data is stored in the mobile device and is sent to the MAVIS subscription manager. The mobile application enables users to create new subscriptions on-the-fly, and supports the dynamically changing needs of system administrators. The MAVIS subscription manager is mainly responsible for storing all the incoming subscriptions from the subscribers. Mobile applications interact with the subscription manager by subscribing for events written in the DSL. The subscription manager can be configured to create groups that contain a set of nodes within the datacenter. This becomes useful when subscribers try to create subscriptions for a large set of machines in the datacenter. The subscription manager is responsible for creating individual subscriptions for each machine in the group.

The subscription manager can which CEP engines will do selective offloading of some subscriptions to the particular target node. The rationale is that all the information for matching the subscription is locally available at that node and the computational resource needed to process the subscription against the metric values is minimal. So, rather than incur the expense of communicating the stream of metric values from the publisher (the target machine) to the intermediate server, MAVIS decides to let the processing happen on the publisher itself. MAVIS can control the resource utilization on the publisher, which is a crucial setting in most systems. It does this by only offloading the easy-to-process subscriptions (such as, min and max) and bounding the total number that will be processed on any target machine.

On performing selective offloading, the corresponding CEP engine maintains a list of the subscriptions sent to the publisher such that they are in sorted order. This ordering will facilitate matching multiple subscriptions at the same time. The subscriptions which are not offloaded are sent to the Streams applications within the CEP engine, so that Streams can use the subscription to processes incoming system metrics.

The MAVIS publisher receives subscription data from the CEP engine during the selective offloading process. The subscription data includes the metric(s), the subscription type, and the threshold value. It maintains a sorted list of the threshold values to facilitate matching multiple subscriptions against a single stream of values for one metric. For example, if there are multiple subscriptions for the CPU load, one $> 50\%$, the second for $> 60\%$, and so on, when the metric value comes in (say at 75%), the publisher does a binary search to determine which subscriptions have matched.

3) Streaming Event Processing

The MAVIS CEP engine is primarily responsible for matching subscriptions and detecting events. The event detection engine is based on IBM's InfoSphere Streams [8] stream processing engine. The use of Streams is motivated by the observation that for many applications of systems management, it is hugely inefficient to store the monitored data in a datastore and then run queries against the datastore. This is because fundamentally the queries are on data as it streams forth from the target machines and the queries are on windows of the data that are constantly sliding. In InfoSphere Streams or Streams, queries are composed of operators and the data flows between these operators. Federations of queries are called *applications*. Whereas individual operators and their compositions cannot be dynamically altered in a running Streams system, applications can be dynamically loaded, unloaded, and rewired through their input and output ports. We create several Streams applications to process the subscriptions from different publishers. In our current implementation, we instantiate multiple Streams applications in each CEP engine. There is one Streams application for each subscription type (thus, 6 in each target end point). The processing of multiple event streams within one Streams application is sequential, which is a limitation of our current implementation, not the Streams framework. The processing of events in different Streams applications happens in parallel.

4) Event Detection

In MAVIS, the process of detecting events and forwarding them to subscribers is split up into four parts – metric collection, subscription matching, selective storage, and selective notifications.

Metric Collection. The MAVIS publisher is responsible for collecting metrics such as CPU, disk space, memory and network statistics. MAVIS can obtain these metrics from an external source such as Nagios [6] or Ganglia [7]. These monitoring tools provide plugins that can be used to obtain the various metrics. We do not make any contributions towards the metric collection, and our solution is agnostic to the metric collection software.

Subscription Matching. Upon obtaining the metrics, if the publisher has any subscriptions to process locally through selective offloading, it does a binary search on the sorted lists using the metric value and passes on the closest matched index to the CEP engine. The MAVIS CEP engine receives metrics from a group of publishers and invokes the Streams application(s) for subscription matching. The CEP engine also processes the results provided by the publisher as a result of the selective offloading. It uses the index returned by the publisher and uses it on the sorted lists to match multiple subscriptions at the same time. The sorted lists also contains information on how to contact the subscriber.

If any of the subscriptions can only be partially matched at a CEP engine — because it is spatial and the group of machines referenced in the subscription spans multiple CEP engines — then the partial match result is passed on to the subscription manager. The subscription manager then strings together the partial matches and creates the complete match, and notifies the subscribers if there is ultimately a match.

Selective Storage. When a subscription is matched, the CEP engine stores the data for *all* the metrics of the associated publisher, so that these metric values can potentially be used for problem diagnosis. We refer to this as MAVIS's *selective storage* component. We maintain a selective storage, so that administrators can manually query the storage when they wish to analyze and diagnose the problem. The mobile application supports a pull functionality for this kind of post-mortem debugging. We do not need to store data if there is no alert since our complex subscription language anyway allows the admins to craft stateful queries. Our current policy is to expire and remove the selectively stored data from the intermediate server after 24 hours.

Selective Notification. Once a subscription is matched, it normally results in an alert, *i.e.*, a notification to the corresponding subscriber(s). MAVIS intermediate server sends an alert to the subscriber the first time it gets matched. If it continues to get matched, we do not send the subsequent alerts. The rationale behind this is that once the administrator is notified that an error exists in the system, she would want to query the selective storage to analyze the problem. She would not like to be bombarded with alerts corresponding to the persistent problem. Instead, we send an alert once the subscription no longer gets matched indicating that the error has been removed from the system.

In order to send notifications to the mobile device, we need to use a push notification service. The CEP engine communicates with the MAVIS subscribers using the Google Cloud Messaging (GCM) push feature. When the mobile application is launched for the first time, it registers with the GCM servers and obtains a unique identifier which enables the device to receive push notifications using the framework. Similar solutions such as Apple Push Notification (APN) service exists for iPhone devices.

V. EVALUATION

In this section, we present the evaluation of MAVIS. We use as a basis for the rules we use in our evaluation, the results of the survey of the system administrators (Section VIII-B). Our evaluation is meant to answer the following questions.

(1) What is the detection latency for MAVIS when there are failures? We measure the time between when the publisher obtains the metric data to when the subscriber on the mobile client receives an alert.

(2) What is the resource utilization of MAVIS as a function of the number of machines being monitored, when there is no failure? We compare the resource utilization and scalability of MAVIS with two comparative systems, aNag and Volley.

(3) What is the maximum number of subscriptions that can be processed on a single CEP engine and how does distribution of CEP engines help in scalability.

A. Experimental Setup

In order to conduct the evaluation of our system, we use monitoring data obtained from the host machines within ITaP, Purdue's central IT organization. ITaP monitored data is available for approximately 3,000 different machines, where each machine has between 6 and 12 metrics being monitored on them. These machines are collectively owned by various faculty members all through the university and correspondingly run a wide variety of scientific workloads, typically in a batch mode. They are centrally managed by ITaP, kept in a few centralized datacenters, and are typically not physically accessible to the researchers running applications on them. The machines and the datacenter are considered to have state-of-the-practice monitoring and management and boast of uptimes of greater than 95% for all the clusters in 2014 [10]. Of these, about 200 machines support virtual machines (VMs) and on an average, each physical machine supports 10 VMs. We obtain 4 hour continuous streams of metric data from the production machines, accumnulated over 1 month in 2014. In our experiments we create a maximum of 4,620 subscriptions per physical machine. We monitored the following 7 metrics – User CPU, System CPU, used disk space, free memory, memory buffer space used, entropy, and ambient temperature. We arrive at this number based on the criteria discussed in Section II. In our experiments we simulate physical machines by executing multiple publishers and have them read ITaP metric values from a file. In reality, the physical machine executes a plugin (such as, a Nagios plugin) to retrieve the metric value online.

B. Experiment 1: MAVIS *Latency of Event Detection*

It is essential that MAVIS subscribers on the mobile devices are able to receive alerts in a timely manner. This involves processing at the publisher, the CEP engine, pushing notifications to the client, and finally receipt of the alarm at the client. In this experiment we measure the end-to-end latency taken by our entire system to deliver a notification to the mobile device. We start the timer when the publisher receives the metric data on the target machine, and stop the timer when the subscriber receives the notification. In order to measure the performance of our system, we vary the number of machines being monitored and inject synthetic failures in the data. Each physical machine has 10 virtual machines. MAVIS monitors the metrics of the physical machines as well as the VMs. In Figure 3 we show how the latency of receiving alerts at the mobile end points varies based on how we utilize the streaming event processing engine. Through the use of IBM Streams, we are able to divide the stream processing among different Streams applications. We divide the work in two ways — (a) splitting up the total number of physical machines monitored among two Streams applications, and (b) splitting up the subscription types for *all* of the monitored physical machines among three Streams applications. Note that a value of 50 physical machines being monitored (X-axis of Figure 3) means there are a total of 550 end points (VMs + physical machines) being monitored. We created approximately 4000 subscriptions(max number of admins*number of metrics per machine*2) per publisher. We injected synthetic failures in a uniform random manner in the data, such that 2% of the overall number of subscriptions per machine would get matched. We believe 2% is a reasonable estimate as failures are rare, and

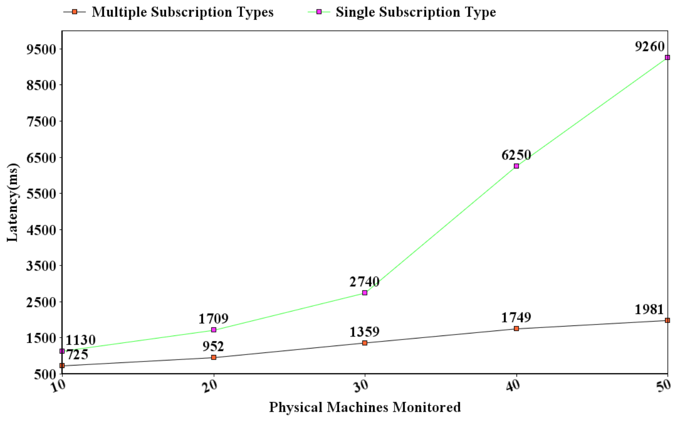


Fig. 3. Average latency for receiving alerts at the MAVIS subscriber (the mobile device) as a function of number of target machines being monitored

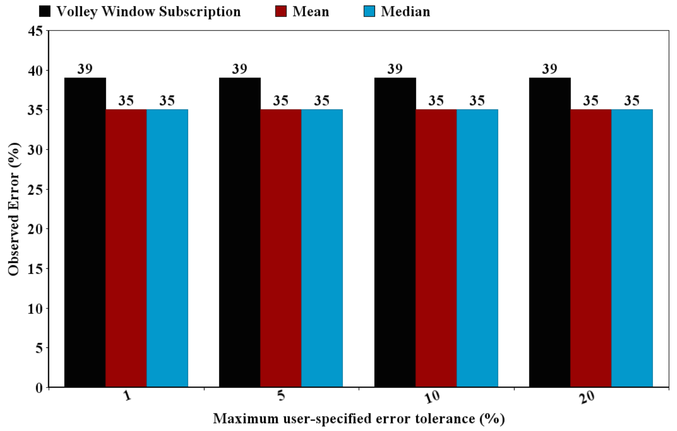


Fig. 4. Error percentage observed on Volley as a function of allowed error percentage

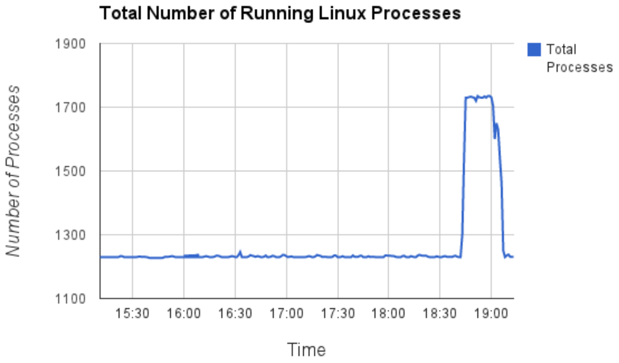


Fig. 5. Part of the ITaP dataset showing the number of running Linux processes as a function of time

our selective notification algorithm limits the number of alerts when a failure is present. For this experiment, publishers read the metric data from the file and send it to the CEP engine every 15 seconds.

We observe the effect of a varying number of machines on the end-to-end latency and the result is shown in Figure 3. The first way of decomposing into Streams applications (by the physical machines) shows that processing all of the subscription types on a single Streams application does not scale well, since the subscription types have to be processed sequentially. Splitting up the subscription types allows each subscription type to be processed in parallel and minimizes the latency. In evaluating the response time of our system, scaling up to a ceiling of 550 target end points created a latency of approximately 2 seconds, which seems tolerable for most system management functions.

C. Experiment 2: Volley Accuracy and Resource Utilization

While researching related work in the field of distributed state monitoring, we encountered Volley [15], a recent solution which performs *all* subscription matching on the datacenter nodes. Volley aims to contain the resource utilization on the datacenter node by utilizing an algorithm which dynamically changes the monitoring frequency for each metric based on how variable the metric is and how close the metric value is to the threshold. While MAVIS tries to match metric values against subscriptions at fixed intervals, Volley aims to reduce the frequency at which this *sampling* is done. However, by reducing the frequency of sampling, Volley can sometime miss events of interest. Volley has a default sampling interval, which is the most frequent rate at which it will do sampling. Volley takes in an argument for the maximum error allowance — the maximum percentage of subscription matches that it can miss — and tries to ensure that the sampling frequency is such that the actual error does not exceed the allowance. Similar to MAVIS, Volley can support local and spatial subscription types. Volley supports a narrow set of local subscription types — basically where it collects a window of metric values and the subscription checks if *all* the values are above (or below) a user-specified threshold value. We approached one of the authors of Volley for the implementation but one was not readily available, so we implemented the design faithfully according to the description in the paper. For this experiment, we set Volley's default sampling interval to the highest granularity for the ITaP data, which is 1 minute.

In our comparison with Volley, we first evaluate if Volley is suitable for the events in systems management, which are typically rare and sometime short-lasting. For this, we set an error threshold (which we vary in the experiment), create a subscription in Volley (we vary the type of subscription among three types), and measure the actual error rate when run on the ITaP dataset. We extend Volley beyond the single type of subscription that it supports, to also support mean and median. Our objective for adding these two subscription types is that they are more stable, *i.e.*, they are less noisy due to fluctuating metric data.

Specifically, we ran Volley against the metric "Total number of processes" in the ITaP dataset (Figure 5). The dataset represents the total number of running processes on a given machine. In the dataset. we observe that initially the data is steady with little variation and then we observe a spike in the total number of running processes. This pattern recurs through the dataset. We observe that these types of metrics are not seen very frequently, but are still present (approximately 5% of the metrics). These spikes are important to detect because they point to anomalies that can, if left unchecked, lead to a cascade of errors. For example, when the queue length for a load balancer shoots up, it starts to drop requests leading to user-visible outages. We find that with this class of metrics, Volley misses the beginning part of the spike, or if the spike is narrow, the entire spike itself. We observe an error percentage of 39%, 35%, and 35% for the Volley standard subscription, mean subscription, and median subscription, irrespective of the specified maximum error allowance. Next, we hypothesize the reason for the large observed error rate in Volley. –

Volley is able to place an upper bound on the mis-detection rate for a given sampling interval based on the assumption that the delta between samples is a time-independent random variable. Chebyshev's inequality is used to compute the upper bound, which allows for the random variable to be of an arbitrary distribution, as long as the mean and variance are known. The mean and variance statistics are allowed to change over time and are updated by computing the statistics over a 1000 sample window. Because these statistics must be computed over a window of time, when the true mean or variance of delta changes very abruptly Volley is not able to correctly place an upper bound on the mis-detection rate. Take for example Figure 5, where there is a very abrupt spike in the metric values around 18:40. Up to this point Volley has computed a mean and variance of *the delta* of the metric as -0.041 and 0.730 respectively, after 18:40 the mean and variance are computed as 0.552 and 4.221. Before 18:40 Volley reduces the sample interval because the very low mean and variance make the metric unlikely to cross the threshold. This results in a long delay to detect the sudden spike at 18:40.

There is another class of metrics which is quite variable, such as the CPU utilization. These kinds of metrics are observed more often among the physical machines at ITaP. In these metrics, the spikes are more frequent, and they are not that high relative to the rest of the measurements. In this class of metrics, we observe that Volley does *not* miss the spikes, so the error rate is small, but it suffers from the other problem that its sampling interval rarely rises above the default interval. As a result, Volley misses the its goal of reducing the monitoring overhead.

CPU Execution Time We also compared Mavis's CPU execution to Volley's by using the ITaP dataset in Figure 5. We vary the number of subscriptions processed on the physical machine, and observe the total execution time (CPU user + CPU system time). We have a ceiling of 4,620 subscriptions since physical machines can be running VMs. The subscriptions used here are the same ones used by Volley. We observed a result that was surprising and counter-intuitive. Figure 6 shows that Mavis's execution time that was faster than Volley's as we varied the number of subscriptions. The reason for this initially surprising result is that Volley uses a backoff parameter immediately after an alert when it does not change the sampling period. With our dataset and the backoff interval from [15], Volley stays at the default sampling interval for a significant amount of time. Volley also treats every subscription independently, regardless of whether they use the same metric or not. As a result, it requires much more computation per subscription, while Mavis amortizes the cost of processing multiple subscriptions through techniques such as the sorted list of thresholds (Section IV-2). These two factors together lead to more efficient CPU usage in Mavis.

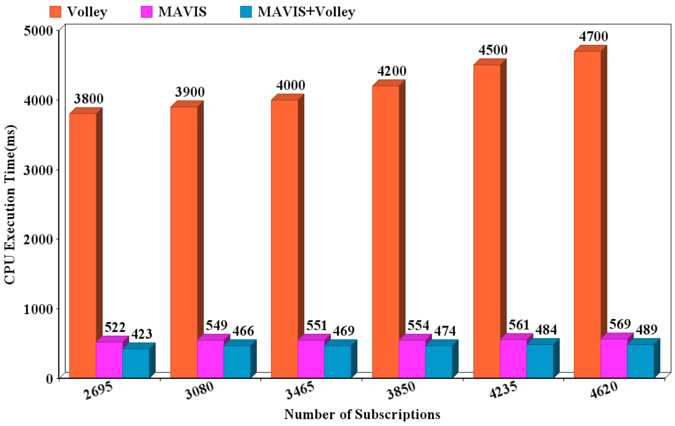


Fig. 6. CPU Execution time of MAVIS, Volley, and MAVIS adapted with Volley's dynamic sampling algorithm

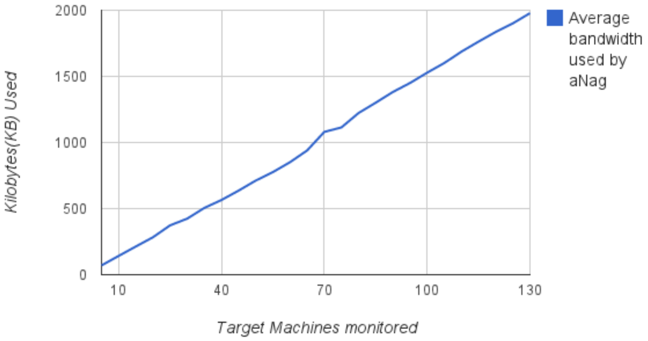


Fig. 7. Bandwidth utilization by aNag as a function of monitored target machines

D. Experiment 3: aNag Bandwidth Utilization

In the process of researching existing mobile management solutions for systems management, we came across aNag, the highest rated system management application on Google Play (rating of 4.8/5). aNag is a mobile application which communicates with the Nagios monitoring software running on the target machines. Nagios only supports simple instantaneous subscriptions. aNag, in order to support richer subscriptions, communicates with the Nagios web interface to retrieve *all* the monitoring data, along with, less critically, the status levels that Nagios calculates (OK, WARNING, or CRITICAL).

We monitored up to 130 target end points (all VMs in this case) using aNag where each target machine was monitoring between 6 and 12 metrics, and there was one subscription per metric. Figure 7 shows the bandwidth utilization of aNag as a function of the number of target machines monitored. Using [12] as a reference for how bandwidth hungry popular applications are, we find that aNag exceeds the average mobile application bandwidth utilization per hour (10.7 MB/hour averaged across the 50 most popular mobile applications) after monitoring just 15 target machines, and surpasses the hungriest bandwidth usage limit (115 MB/hour for the hungriest of these 50 applications) while monitoring just 130 target machines. aNag has to retrieve all the monitoring data from the target machines, regardless of the status of the Nagios subscriptions, and filters them at the mobile application. Hence, we observe a large bandwidth utilization at the mobile device running aNag. Furthermore, due to the large number of network connections made by aNag, the mobile application utilizes an excessive number of threads and runs out of virtual memory for creating additional threads when monitoring more than 597 physical machines. We discuss this result in the Appendix.

MAVIS's utilization of a CEP engine to deliver alerts and its selective notification algorithm prevents the mobile application from utilizing excessive bandwidth. Notifications are only received when a new failure arises in one of the target machines. A failure is characterized by the matching of a subscription, and we define a new failure as a subscription that has not been continuously matched, and has not been recently matched. Our selective notification algorithm ensures that alerts are only sent when a new failure is observed. The bandwidth utilized on sending a push notification is approximately 500 bytes, since we only send information regarding the threshold that matched the subscription, and the time at which it occurred. Administrators can also choose to query our selective storage component to analyze the failure. Querying the metrics for a VM and physical machine over the past 10 minutes, consumes 4,262 bytes. As a result, MAVIS subscribers consume nearly 4KB of bandwidth when a new failure is observed.

E. Experiment 4: Scalability of MAVIS

In addition to the efficient processing of complex events, MAVIS also has the ability to spawn multiple CEP engines and distribute them to multiple physical intermediate server machines, if the load on one CEP engine becomes too high. In the default configuration, each CEP engine has 6 Streams applications, one for each of the 6 types of subscriptions — mean, median, standard deviation, variance, instantaneous min, and instantaneous max. If the number of machines being processed by one CEP engine becomes too high, then another CEP engine is created which handles a disjoint set of machines. In this experiment, we see the effect of this kind of distribution. For this experiment, publishers read the metric data from the file and send it to the CEP engine every 15 seconds.

The resource that becomes the determining factor for distributing the load is the CPU utilization. We set a policy that an intermediate server machine should not have a CPU utilization exceeding 90%. We find from our experimental result, shown in Figure 8, that this water mark is reached with 50 physical machines. Each physical machine corresponds to 4,620 subscriptions (11 target end points/machine * 7 metrics/target end point * 2 subscriptions/metric/sysadmin * 30 sysadmins). When the high water mark is reached, due to the distribution, a second CEP engine is spawned and the CPU utilization on the first server machine comes down to about 49% with each of the two intermediate server machines handling 30 physical machines. This experiment highlights the importance of distribution of CEP engines, a factor that has not been considered in much of the prior work [20], [15]. The experiment demonstrates the currently implemented strategy of dividing the machines to be monitored equally among the different CEP engines. More sophisticated strategies where the division is proportional to the amount of activity, number of VMs, etc. can also be implemented.

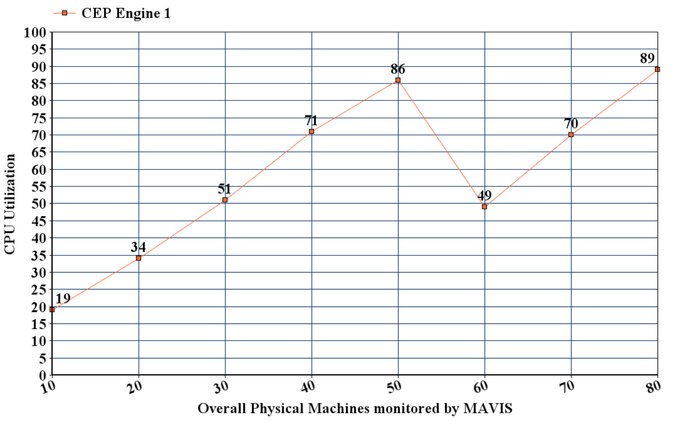


Fig. 8.   CPU Utilization of a CEP Engine as a function of the overall physical machines monitored by MAVIS

We also see the need for selective offloading — a single CEP engine can monitor a larger number of physical machines by offloading subscriptions for instantaneous min and max to the target physical machines. With this selective offloading, each CEP engine can support 10 additional physical machines, *i.e.*, 46,200 additional subscriptions.

VI. RELATED WORK

Conventional monitoring of systems and networks

There is a well established base of technologies for monitoring computer systems and networks for the purpose of early detection of problems. A number of the most popular monitoring systems are available as open source software, such as Nagios [6], Ganglia [7], and OpenNMS [17]. All these monitoring systems periodically probe the compute and network elements under their purview for a set of metrics or conditions and report anomalies to their users, typically system administrators, through web interfaces, email, or text messages. Web interfaces also allow viewing and analysis of historical metric data, which can reveal problems not detected through the periodic probes. These tools are generally quite flexible and configurable, and thus continue to find wide acceptance. However, the ever-increasing scale of datacenters, new workloads and application architectures, advances in computer system architecture (*e.g.*, GPUs, FPGAs, other accelerators), and new failure and attack modes present new challenges that demand continuous innovation in monitoring solutions.

Volley is a recent effort to reduce monitoring overheads on target systems [15]. Monitoring overhead can be a significant issue when metric collection requires significant computational resources, *e.g.*, deep packet inspection. Volley addresses this issue by dynamically adjusting the sampling period for metrics based on the probability of an event being detected. We have already discussed the protocol details during its comparative evaluation with MAVIS. Qualitatively, MAVIS differs from Volley primarily in its support for complex event processing, which allows more sophisticated and focused event detection. MAVIS provides advanced support for mobile monitoring clients, a feature absent in Volley.

HOLMES is one of the earliest attempts to employ a complex event processing engine in a monitoring system [20]. It uses Message Oriented Middleware (MOM) for all communications with target machines and between its processing modules. While HOLMES' CEP component does allow for complex event detection, all events are funnelled through a single instance of this CEP engine, and this could limit scalability. We address this concern in MAVIS by structuring CEP processing using a distributed network of CEP engines. HOLMES also provides no explicit support for mobile clients.

In [14], the authors introduce the idea of a distributed CEP architecture for system monitoring and evaluate one centralized and two notional distributed architectures. However, there is not a great level of technical detail about how to distribute parts of the CEP, what subscriptions to support where, and what communication primitive to use between any two parts of the CEP. We answer all of these points in the current work.

Mobile monitoring solutions

The tremendous growth in the capabilities and numbers of mobile devices is one of the technological changes driving the need for new solutions to system and network monitoring. Some early efforts to address this need are mobile clients for Nagios such as aNag, iNag, Mobile Admin, and TeenyNagios [16]. A subset of the authors of this paper were responsible for developing a solution called IBM Remote [18], which works with IBM Blade Servers. While features and functions vary somewhat across these applications, all essentially pull data directly from the Nagios (or some other custom) datastore for display on a smart phone or tablet. None of them perform the aggregation and alert generation functionality as MAVIS intermediate server does.

A more advanced approach to mobile monitoring is available in the Zabbix enterprise monitoring system and its mobile frontend called Zax [11]. Zabbix is a lightweight monitoring application that is able to manage thousands of items with a single-server installation. While similar in architecture to MAVIS, Zabbix has several drawbacks. The push notification to the mobile application contains all the alert logs, without the ability to select what context information is received at the device. This can adversely impact the bandwidth consumption of the mobile device. To update the alert information involves making changes at the target end hosts through a web interface, as opposed to our system's subscription mechanism. In MAVIS, the interests can be periodically changed by processing a new subscription, but such an evolution is not possible in Zabbix. Zabbix does not provide a mechanism to compress multiple spatial or temporal rules, say from different system administrators. This can lead to a bloat in the number of "service checks" that need to be executed on the target end hosts, with an undesirable increase in load on the target end hosts.

Subscription languages

In MAVIS, we have developed a rich subscription language suited to systems management tasks. There is a long and rich history of rich subscription languages and efficient runtimes for processing them. Content-based publish-subscribe (CBPS) systems, such as [2], [3], support atomic subscriptions over key-value pairs and conjunction or disjunction of atomic subscription units. These subscriptions take wildcards and some aggregation operators. But since these are not specifically geared toward systems management tasks, some of the operators crucial for systems management are either not present, *e.g.*, aggregation across a subnet of machines (in most cases) or are not efficiently implemented, *e.g.*, they are implemented in a centralized manner [13]. However, these systems have very efficient algorithms for distributed filtering and routing of events through their broker network (a simplified version of the CEP engine) and can scale to a large number of subscribers. These are *not* distinguishing points of MAVIS.

VII. CONCLUSION

The ubiquitous availability of mobile phones and widespread network connectivity have led to a demand for real-time information about the state of data centers. System administrators would like access to timely alerts and richer context information on the mobile devices while minimizing the data monitoring overhead on data center host machines, minimizing resource consumption on the mobile devices, and meeting the dynamically changing needs of system administrators. These requirements motivated our design of MAVIS, a framework that facilitates managing datacenter activities from mobile devices by implementing a rich domain specific language and a scalable complex event processing architecture by integrating it with three novel components. Our evaluation shows that the MAVIS mobile application can monitor datacenter activities in a resource-efficient manner. We can manage the activities of 3,000 physical machines within our datacenter by utilizing 6 CEP engines. A single instance of the MAVIS CEP engine running on one machine can handle 231K subscriptions. We perform quantitative comparison with a state-of-the-art monitoring solution, Volley, and find that it misses many events of interest, while the leading mobile systems management app, aNag, consumes 2 MB/s for managing just 130 machines.

REFERENCES

[1] S. Bagchi, F. Arshad, J. Rellermeyer, T. Osiecki, M. Kistler, and A. Gheith. Lilliput meets brobdingnagian: Data center systems management through mobile devices. In *The Third International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV)*, pages 1–6. IEEE, 2013.

[2] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 437–446. IEEE, 2005.

[3] S. Bhola, R. Strom, S. Bagchi, Y. Zhao, and J. Auerbach. Exactly-once delivery in a content-based publish-subscribe system. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 7–16. IEEE, 2002.

[4] Cisco Systems. Meraki Systems Manager. https://play.google.com/store/apps/details?id=com.meraki.sm.

[5] Damien Degois. aNag. https://play.google.com/store/apps/details?id=info.degois.damien.android.aNag.

[6] Ethan Galstad. Nagios - The Industry Standard in IT Infrastructure Monitoring. http://www.nagios.org/.

[7] Ganglia Community. Ganglia Monitoring System. http://ganglia.sourceforge.net/.

[8] IBM. Big Data: Stream Computing. http://www.ibm.com/developerworks/bigdata/stream.html.

[9] iNag2. iNag2 Push Notifications. https://www.inag2.com/2013/a-note-about-push-alerts/.

[10] Information Technology at Purdue (ITaP). Community Clusters. https://www.rcac.purdue.edu/services/communityclusters/.

[11] inovex Gmbh. ZAX Zabbix Systems Monitoring. https://play.google.com/store/apps/details?id=com.inovex.zabbixmobile.

[12] Kevin Tofel. Data Hungry Mobile Apps Eating into Bandwidth Use. http://gigaom.com/2011/05/19/data-hungry-mobile-apps-eating-into-bandwidth-use/.

[13] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, pages 249–269. Springer, 2005.

[14] A. Mdhaffar, R. B. Halima, M. Jmaiel, and B. Freisleben. A dynamic complex event processing architecture for cloud monitoring and analysis. In *5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, volume 2, pages 270–275. IEEE, 2013.

[15] S. Meng, A. K. Iyengar, I. M. Rouvellou, and L. Liu. Volley: Violation likelihood based state monitoring for datacenters. In *IEEE 33rd International Conference on Distributed Computing Systems (ICDCS)*, pages 1–10. IEEE, 2013.

[16] Nagios Project. Nagios Mobile Device Interfaces. http://exchange.nagios.org/directory/Addons/Frontends-GUIs-and-CLIs/Mobile-Device-Interfaces#/.

[17] OpenNMS Project. OpenNMS: Enterprise grade network management application platform. http://www.opennms.org/.

[18] J. S. Rellermeyer, T. H. Osiecki, E. A. Holloway, P. J. Bohrer, and M. Kistler. System management with ibm mobile systems remote: A question of power and scale. In *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*, pages 294–299. IEEE, 2012.

[19] Sonian. Sensu — Sonian. http://sonian.com/about/sensu/.

[20] P. H. d. S. Teixeira, R. G. Clemente, R. A. Kaiser, and D. A. Vieira Jr. Holmes: an event-driven solution to monitor data centers through continuous queries and machine learning. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 216–221. ACM, 2010.

VIII. APPENDIX

A. *Futher rationale for our Approach*

We started developing MAVIS with the goal of managing datacenter failures through mobile devices. We realized that there was a requirement for an intermediate server in order to facilitate communication with mobile end points. Streaming event processing engines have been used in existing desktop/web-based monitoring of datacenters. However, it can be especially useful for mobile monitoring because we cannot afford to have all the data be brought back to the mobile device and run complicated, ad-hoc scripts on the mobile device to make sense of the huge volume of data. In order to maximize the potential of our complex event processing engine, we combine three novel components to enhance the efficiency and scalability of our solution. These are namely selective storage, selective offloading, and selective notifications. These components could be easily used for desktop monitoring, but their importance is much more significant in the context of datacenter management through mobile devices. For example, selectively sending push notifications to mobile devices prevents users from observing a large number of alerts on such a small device and also contains the bandwidth of the device. However, this is not issue in desktop monitoring.

The DSL we support allows a subscriber to specify rich queries that depend on a lot of satte from a lot of different machines in the datacenter. This relieves mobile devices from having to bring all the state into itself and then do local processing. Thus it turned out that the rich DSL for systems management subscriptions, while valuable even for desktop monitoring, is of particular value for mobile management of datacenters.

MAVIS has the ability to support both desktop monitoring and mobile management of datacenters. The components within MAVIS enhance the systems efficiency in both scenarios, but the need is more evident for mobile management of datacenters. As a result we focused on developing the susbcribers to be mobile applications, but they could also be developed as desktop clients.

B. *System Administrator Survey*

To inform the design of our system monitoring application, we conducted a survey of system administrators at the central IT organization of a tier-1 research university and at a large industrial research lab. The purpose of the survey was to identify conditions that they would find most useful in detecting potential failures. Our survey asked system administrators to rate the usefulness of a variety of instantaneous, temporal, and spatial rules for identifying potential failures. We also requested quantitative information on limits and thresholds for any rule that the respondent indicated were at least somewhat useful. The anonymous survey is in fact available for the reader to see or participate in at https://purdue.qualtrics.com/SE/?SID=SV_exGUHOkGSdonTYF.

We received 18 complete responses to our survey. While this number is too small to draw statistically significant conclusions, we feel that the data is still helpful as a guide in selecting the health checks to perform on the systems monitored by our application. The seven rules (out of a total of 23) that received the highest average rating for usefulness in our survey were:
(i) Alert (within a few seconds) if a particular server gets switched off. (Instantaneous rule)
(ii) Temperature exceeds a threshold at any point in time. (Instantaneous rule)
(iii) Average Round Trip Time (RTT) to a server over the past N minutes exceeds X ms. (Temporal rule)

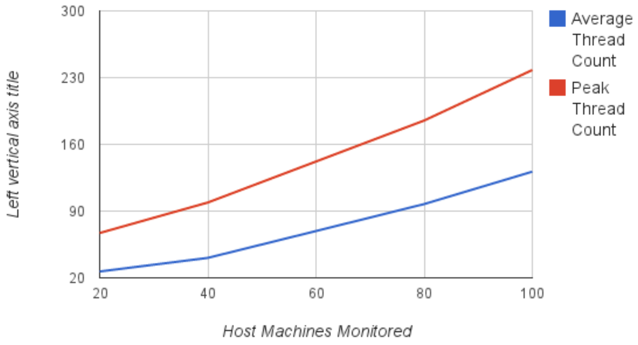


Fig. 9. Utilization of the number of threads by aNag (the key constrained resource) as a function of the number of monitored target VMs.

(iv) Average server temperature for the past N minutes is above X degrees. (Temporal rule)
(v) Alert (within a few seconds) if the management module on the Blade chassis becomes unreachable. (Instantaneous rule)
(vi) Average disk space free for the past N minutes is less than X per cent. (Temporal rule)
(vii) The amount of free memory for each server is less than X MB (Instantaneous rule)

We observe that these top seven conditions are split roughly between instantaneous and temporal conditions, which supports our basic assumption that the infrastructure must support analysis of temporal phenomena. The most highly rated spatial condition - average temperature across as set of servers - falls just below these top six in the ranking, so evidence for spatial analysis is weaker but still present.

Also significant is the high importance given to alerts delivered within a few seconds of a server becoming unavailable. This supports our view that the infrastructure used for our monitoring application must be capable of identifying and then delivering the notification of an event in the timescale of seconds.

Finally, the detailed responses indicate a significant diversity of limits and thresholds that system administrators would like to employ in their monitoring tasks. For example, for Average Round Trip Time (RTT) to a server over the past N minutes exceeds X ms., respondents recommended values for N as low as 1 minute and as high as 20 minutes, and values for X ranged from 30 ms to 400 ms. This supports our view that the infrastructure should support a varied, even personalized, set of rules for specifying events of interest.

C. Thread Utilization

In order to measure the scalabilty of aNag we monitored up to 100 target end points, where each target machine had between 6 and 12 Nagios service checks. Figure 9 shows the increase in average and peak thread utilization by aNag. We noticed when monitoring more than 597 target machines the thread utilization increased to an extent where the mobile device could not create any more threads and returned an exception due to limited virtual memory, causing the application to crash. This problem arises when we setup Nagios instances on each target machine, and each target machine has its own web interface for managing subscriptions. Nagios can also be setup such that each node sends its subscription data to a gateway machine which displays the subscriptions for all datacenter nodes in a central web interface. When each target machine has a web interface, aNag must create threads to establish a network connection with each target machine. We find from Figure 9 that the average number of threads per machine that aNag uses is approximately 1, while the peak is greater than 2. Naturally, it is the peak usage that causes a resource exhaustion problem. MAVIS does not experience this drawback, since the mobile application waits for the CEP engine to send alerts and does not create threads per target end host.

D. Grammar for the Subscription DSL

Here we present the formal grammar for the Domain Specific Language (DSL) in which our subscriptions are expressed.

⟨*top-level-subscription*⟩ ::= ⟨*subscription*⟩ [⟨*spatial*⟩] [⟨*temporal*⟩]

⟨*subscription*⟩ ::= ⟨*subscription*⟩ AND ⟨*subscription*⟩
| ⟨*subscription*⟩ OR ⟨*subscription*⟩
| ⟨*basic-subscription*⟩

⟨*basic-subscription*⟩ ::= ⟨*aggregator*⟩ ⟨*metric*⟩ ’lt’ ’le’ ’eq’ ’gt’ ’ge’ ⟨*value*⟩

⟨*aggregator*⟩ ::= MEAN
| MEDIAN
| STDEV
| VARIANCE
| MAX
| MIN

⟨*spatial*⟩ ::= ⟨*machine-id*⟩
| ⟨*machine-id*⟩+

⟨*temporal*⟩ ::= ’(’ ⟨*time-value-start*⟩ ’,’ ⟨*time-value-end*⟩ ’)’