MEDAL: A Middleware for Complex Event Detection and Analytics from Smartphones

[Research Paper]

Benjamin Kobin[*]
Purdue University
bkobin@purdue.edu

Raghavendran Shankar[*]
Purdue University
rshankar@purdue.edu

Saurabh Bagchi
Purdue University
IBM Research
sbagchi@purdue.edu

Michael Kistler
IBM Research
mkistler@us.ibm.com

Jan S. Rellermeyer
IBM Research
rellermeyer@us.ibm.com

ABSTRACT

The ubiquitous availability of mobile phones and widespread network connectivity have led to a demand for real-time information about the state of computing devices and infrastructure such as data centers. One example where this is highlighted is in system management in large commercial data centers. System administrators have used simple text or email-based status notification about machines and network equipment state changes for a long time. However, in the era of smartphones this is no longer sufficient as the system administrator is expected to immediately initiate problem analysis and mitigation from the mobile device. Therefore, they need access to a much richer data set and far better usability of tools available on the mobile devices. This poses several technical challenges — timely processing of monitored data and generation of alerts to mobile devices, efficiently storing prior alert data to provide system administrators access to the problem context, minimizing the resource consumption on the mobile devices, and meeting the dynamically changing needs of system administrators.

Existing solutions for mobile system management painfully expose their inability to address these challenges. This motivated our design of MEDAL, a middleware platform for evaluating high-volume data streams, generating alerts based on dynamic rule-based subscriptions, retaining sufficient data around detected events for effective problem analysis, and efficiently disseminating this content to subscribed mobile devices. Our evaluation shows that mobile system management solutions can be much more efficiently implemented using our MEDAL middleware and the scalability and resource utilization on the mobile devices are significantly better than existing state-of-the-art mobile systems management applications like aNag.

Keywords

server management; systems management; mobile devices; streaming query processing

1. INTRODUCTION

The pervasive collection and availability of data have led to a steep increase in demand for real-time information and insights which in turn led to even more data collection. In the era of plentiful data the raw data itself can no longer be assimilated by humans in a timely manner, it is the insights derived from it that are useful to humans. At the same time the ubiquitous use of mobile phones and widespread network connectivity has irreversibly changed the way how we consume this data. Previously it was an off-line task, often with significant manual intervention, to analyze the data and detect patterns in it. Now, the expectation has shifted towards automatically ingesting and analyzing the data and detecting complex events to gather insights and subsequently send notifications to the mobile device in close to real-time. However, mere detection of an event of interest is rarely sufficient to the user for acting on the event. The recipient often requires context, possibly by acquiring additional data, to support the decision process and subsequent action. Thus, the workflow has turned into an online processs, with the expectation that the system will enable the user to act on the event in near real-time.

One example where this trend is highlighted is in system management of large commercial data centers. System administrators[1] have used simple text or email-based status notification about machines and network equipment state changes for a long time. However, in the era of smartphones this is no longer sufficient as the system administrator is expected to immediately initiate problem analysis and mitigation from the mobile device. This is an impedance mismatch as the data available from the system is increasing with every generation of servers and can easily comprise of hundreds of metrics per machine or network box. Further, the sizes of data centers are increasing and therefore, the number of machines under the purview of a system administrator has been increasing. Therefore, system administrators today need access to far more usable tools available on the mobile devices. This poses several technical challenges—timely processing of monitored data and generation of alerts to mobile devices, efficiently storing prior alert data to provide system administrators access to the context for troubleshooting, minimizing the resource consumption on the mobile devices, and meeting the dynamically changing needs of system administrators.

Analytics typically involves full warehousing of the data and subsequent analysis of the entire stored body of data. This is what is colloquially referred to as *big data processing*. When the volume of incoming data is high, this quickly exceeds the capacity of com-

[*]The first two authors have contributed equally to the work.

[1]We use the terms "system administrator" and "user" synonymously in this paper, which is different from common usage. This is because in our system, the target users are the system administrators.

puter infrastructure and only few organizations in the world have the resources to apply a full warehousing strategy to large-scale problems. In the context of our example of system management, you clearly cannot afford running another datacenter only to monitor your first datacenter. Furthermore, most analytics engines run as batch jobs and not nearly in real-time.

Complex event detection in a high volume of streaming data is mature technology. For example, it is the subject of several popular products, such as, IBM's Infosphere Streams [9], SAS' Event Stream Processing Engine [**?**], and SAP's Event Stream Processor [**?**]. Stream event processing typically treats the data as ephemeral, i.e., after it has left the processing pipeline it is discarded. This, however, means that the system maintains no lineage of data that led to the event and the receiver of the event notification can only operate on the compressed insight but not reason about the cause of the event. This is a critical shortcoming in many cases, including for system management, where the context is needed to interpret and act on the detected event.

These shortcomings motivated our design of the MEDAL (**M**iddleware for **E**vent **D**etection and **A**na**L**ytics) middleware for evaluating high-volume data streams, generating alerts based on dynamic rule-based subscriptions, retaining sufficient data around detected events for effective problem analysis, and efficiently disseminating this content to subscribed mobile devices. MEDAL combines the properties of a stream processor with those of a data store and of a publish-subscribe system geared to resource-constrained mobile devices as the subscribed clients. We have successfully used the MEDAL framework to build an application for system management through the mobile device, called MAVIS (**M**obile **A**lert **V**ia an **I**ntermediate **S**erver). MAVIS not only scales significantly better than existing state-of-the-art mobile systems management applications but also gives the system administrator access to richer data while using only a single server machine as infrastructure. It is particularly suited to mobile devices because it consumes less resources, particularly, bandwidth and compute power, and correspondingly improves the battery life.

Our entire framework comprises three entities — target end hosts with metric collection software executing on them, an intermediate server, and the mobile device end points. The intermediate server includes (a) an efficient streaming event processing engine that can run queries efficiently on streaming monitored data and generate alerts upon a match, (b) a no-SQL data store for storing the raw data that can provide context for subsequent troubleshooting, and (c) a store of the subscriptions from the various system administrators. MAVIS has the ability to allow multiple system administrators to express different but overlapping interests among the machines in the data center. For example, 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. On receiving the subscriptions, MAVIS synthesizes these interests, and creates optimized streaming event processing pipelines for them, such that, the latency of the overall processing is kept low. One key benefit of using MAVIS over existing mobile system management solution (such as, aNag, which we compare MAVIS to in a quantitative manner) is that we reduce the bandwidth consumption between the intermediate server and the mobile device, at the expense of higher bandwidth consumption between the target end hosts and the intermediate server. This is advantageous because communication and data processing on the mobile device is expensive and bandwidth between the intermediate server and the mobile device, typically on the cellular network, is constrained.

In this work, we make the following contributions:

1. A framework that allows system administrators to efficiently monitor commercial data center activities using resource constrained mobile devices.

2. An evaluation of the resource utilization, scalability, and accuracy of our solution compared to the current state-of-the-art mobile monitoring solution. For comparison, we use the highest rated application from Google Play Store, called aNag, which is billed as a Nagios (and Icinga) client for Android devices.

3. We describe a case study of how our framework has been utilized to solve system management problems within the servers managed by Information Technology at Purdue (ITaP).

2. MEDAL: DESIGN AND IMPLEMENTATION

MEDAL combines three tasks in one middleware system: managing subscriptions and disseminating content, analyzing large volumes of streaming data and detecting the events that match current subscriptions, and retaining and storing data around detected events for the subscribers to retrieve on demand. These tasks are not orthogonal but complement each others, an important insight that helped us creating 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. The specific event of interest that is matched indicates what kind of data to retain. For example, consider the event that the average temperature of a rack of machines exceeds a threshold value. In that case, the data for the temperature at all the machines in the rack must be retained.

2.1 Subscriptions

Mobile applications interact with the MEDAL server by subscribing for events by posting rules that can be expressed in a domain-specific language. These subscriptions are turned into matching patterns for event detection as well as used for disseminating notifications about matches back to the subscriber. We offer a rich subscription language in which users are able to express their queries of interest. These queries can also include a spatial component and a temporal component. In the context of our motivating example, this would be alerts across different machines and across past time windows. On receiving a subscription request, MEDAL takes the subscription and subscriber info and stores them into a MongoDB database. We use the subscription data to generate rules which will allow MEDAL to detect events and trigger alerts.

Our domain specific subscription language builds on the standard features found in existing content-based pub-sub systems. For background, the standard subscription looks as follows:

$\langle \textit{subscription} \rangle$ ::= $\langle \textit{filter} \rangle$ ('OR' $\langle \textit{filter} \rangle$)*

$\langle \textit{filter} \rangle$ ::= ($\langle \textit{key} \rangle$ $\langle \textit{comparison operator} \rangle$ $\langle \textit{constant} \rangle$)+

The AND operator is implied among the different terms of F1.
Example: F1: Airline = AA, Destination = AUS, Price $<$ \$500
F2: Airline = SW, Destination = AUS, Price $<$ \$600
Subscription = F1 OR F2

For our purposes, we augment this with aggregation operators and these can work both over time ranges and spatial ranges.

⟨*aggregation statement*⟩ ::= ⟨*aggregation operator*⟩ (⟨*range in time or sp*
⟨*enumeration in time or space*⟩) ⟨*metric*⟩

⟨*filter*⟩ ::= ⟨*aggregation statement*⟩ ⟨*comparison operator*⟩ (⟨*constant*⟩
| ⟨*aggregation statement*⟩)

Example:
1. MEAN (t = 5 mins) (CPU load) > 50%
2. RANGE (IP address = a, b, c, d) (Temperature) < 10 degrees

Various aggregation operators are supported, some are more efficiently implementable than others — mean is, creating a Normal distribution is not. For spatial aggregation, there should be a way to identify the machines, IP address being the obvious choice. For temporal aggregation, drifts in clocks among the different end points will cause problem but that problem is expected to be resolved prior to storing the information in the data store, such as, through the use of NTP [5].

We note that two kinds of aggregation operators have distinctive use for systems management. One is the delta operator: $\frac{\partial y}{\partial t}$ or $\frac{\partial y}{\partial x}$, where y is the metric that we are monitoring (fan speed, CPU load, number of processes, etc.), t is the time dimension, and x is the spatial dimension. The second kind of aggregation operator is the ordinal position of a machine with respect to some metric among a peer group (which can be defined in one of a variety of ways). Thus, an alert can be triggered if a machine was running some application with the shortest completion time (ordinal position 1) and becomes one that has the longest completion time. Both these feed into rules where the system administrator is not aware of the absolute value of a metric corresponding to normal operation, but is able to express abnormality in relative terms.

2.2 Event Detection

The MEDAL server consumes data either by polling it from data sources or by receiving streams of data from external data sources. It then continuously applies the matching rules derived from the subscriptions to detect events. The event detection engine is based on IBM's InfoSphere Streams [**?**], the stream processing engine. The use of Streams is motivated by the observation that for many applications storing all the data is not feasible; rather, the data needs to be consumed in a streaming manner and events of interest need to be determined from the stream. This is exactly the fundamental design principle of stream processing engines. In contrast to traditional database technology where the data is persistent and the queries volatile, stream processing engines consider data to be volatile and queries to be continuous and persistent [1].

In InfoSphere Streams, queries are composed of operators and the data flow between them. 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.

Therefore, by default every subscription in MEDAL is compiled into a Streams application which then continuously evaluates the query to detect events. When the subscription is terminated or expires, the application can be unloaded. These per-user applications all consume the identical data stream which they are wired to through the input port, which gets its information from the target end hosts. Furthermore, every application in MEDAL has an output port through which they emit detected events.

In the event that multiple subscriptions are interested in the same or sufficiently similar events, MEDAL can perform de-duplication and factor out the common query into a separate, shared Streams application. In the context system administration, if system administrator A is interested in receiving alerts if the number of pro-

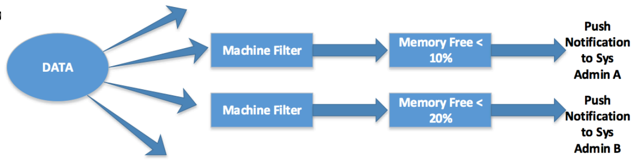


Figure 1: Redundant subscription processing using streams

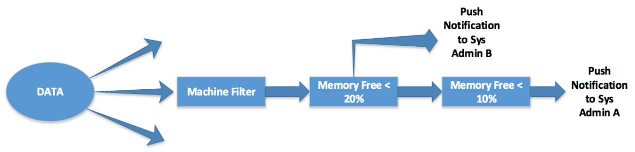

Push Notification to Sys Admin B
DATA
Machine Filter
Memory Free < 20%
Memory Free < 10%
Push Notification to Sys Admin A

Figure 2: Optimized subscription processing using streams

cesses exceeds 100 and additionally system administrator B wishes to receive an alert if the number of processes exceeds 150, rather than realizing the two subscriptions independently as in Figure 1 MEDAL optimizes by cascading the two subscriptions and rewrites them as shown in Figure 2.

In our current implementation, when a subscription is created from a request from a mobile device, the appropriate Streams query is statically created inside the Streams framework. In future work, we plan on developing a technique to dynamically modify the flow our monitored data takes in our Streams applications. This corresponds to the use case that the user changes her subscription, but only slightly. This should be handled more efficiently than unloading a Streams application and loading a new Streams application.

2.3 Selective Data Store

As data streams through the system, it is stored in a NoSQL data store that can be queried if an alert is triggered in the future. We choose to use a NoSQL data store (currently MongoDB) because the nature of the monitoring information is such that strict consistency and transactional semantics are not needed and we can benefit from the performance advantage of a NoSQL data store over a relational database. In order to strike a balance between conserving disk space and keeping useful information available, the expiration time for reading information stored in this database changes dynamically. For instance, the default expiration period for our mobile system management application is 24 hours. However, if an alert of any type is triggered on a machine, all readings related to the machine are updated to expire after one week. Keeping the recorded data in the intermediate server ensures the data will not expire before it can be analyzed, while also allowing ample time for the data to be stored permanently if desired. Keeping data from all monitored readings from a given machine, instead of simply the readings which triggered the alert, ensures possible correlation between readings can be analyzed, even if the relation is not evident to those managing the server. An example of this is provided by a real failure case that we have observed—an increase in the temperature profile of one machine is caused by a bug in the load balancer that was preferentially imposing load on this one machine. This could be unearthed only by looking at load information across the machines in that cluster, though the alert was flagged by the temperature profile on this one machine violating a threshold.

2.4 Notification

In the scenario where an event is detected, MEDAL sends a push notification to the end user application running on the mobile device. On receiving such alerts the end user has the option to obtain prior raw data for the service and visualize the data on the mobile device. Obtaining the prior raw data involves querying the NoSQL data store. In our current implementation, we use the Google Cloud Messaging (GCM) [7] software as the notification service. This has the advantage that it comes in-built with Android and supports disconnected operation. To expand on this, if a mobile device is not connected to the network, GCM will buffer messages and push them out when the device re-connects. A similar service exists for Apple devices and is known as Apple Push Notification service (APNs).

MEDAL stores information about all alerts so that users can use the log of data to identify potential causes of the alert trigger. In the context of system administratiotn, administrators can use the log of data to identify the cause of a failure and initiate mitigation actions from their mobile devices, such as rebooting a server. In other scenarios such as stock trading, a trend of the stock prices can help users decide if they should buy or sell certain stocks.

3. SYSTEM ADMINISTRATION WITH MOBILE DEVICES

Monitoring commercial data centers is a challenging task for system administrators. It is essential that they are able to monitor the various host machines and identify when a fault occurs on these machines. In commercial data centers the number of host machines ranges upward from a few thousand machines. The traditional solution is to have monitoring software tools installed on the monitored target end hosts and have administrators use desktop-class machines to view the monitored data. An example of such a monitoring tool is Nagios [4].

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 monitor 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.

3.1 Requirements

While there exist mobile applications that allow system administrators to monitor physical target machines, they have a variety of limitations. As a growing number of server management professionals begin to use mobile devices as an invaluable tool for service 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 monitoring solutions cause high traffic between end users' mobile devices and the monitored host machines within the data center. It is also common for a group of IT professionals to have different, but partially overlapping degrees of interest regarding which data or which machines to monitor [2]. However, current mobile monitoring solutions do not support this use case. In fact, they are built essentially as single-user applications for the mobile device. By a detailed investigation of existing needs, in conversation with system administrators at an academic institution and an industrial research organization, we outline the following as the primary requirements from a mobile systems monitoring solution:

1. The solution should allow the user to receive timely data center monitoring alerts on a mobile device, provided the user is connected to the network. If the user is not connected to the network, an asynchronous notification should be delivered when he/she does connect.

2. It should be scalable to at least a few thousands of end hosts for targeting reasonable-sized commercial data centers. The solution should also be parsimonious in the resource usage at the mobile devices; the most relevant resources being wireless bandwidth, computation, and resultantly, battery life.

3. The solution should allow administrators to create on-the-fly subscriptions, which indicate what events they are interested in, without having to modify individual configuration files on target machines. It should allow efficient processing of overlapping interests from multiple administrators without a linear (in terms of the number of system administrators) increase in the load on the target machines.

4. It should provide system administrators the context information for an alert. For instance, logs are important for providing the context for the alert and it has often been seen that an error may lay latent in the system for a length of time before manifesting itself as a failure, and a resultant alert. The context information may include the load on the various resources of the machine (CPU, memory, memory bandwidth, storage, etc.), the pattern of interaction of the machine with other machines, the user activity on the host(number of users logged in, number of users with authenticated sessions, etc.).

3.2 Current Monitoring Solutions

In the interest of clarity, we describe this through the example of a widely used, open source monitoring system called Nagios. Nagios can be used to monitor all the hosts and network equipment and services one is interested in. It can send alert when things go wrong and again when they get better. It popularized the centralized polling model, which is by far the most popular model today. In this, there is a central Nagios server which ingests monitoring information from a host of target end hosts, each of which is running some Nagios monitoring code. Under the hood, Nagios is really just a special-purpose scheduling and notification engine. By itself, it cannot monitor anything. All it can do is schedule the execution of little programs referred to as plug-ins and take action based on their output. Nagios plug-ins return one of four states: 0 for "OK", 1 for "Warning", 2 for "Critical", and 3 for "Unknown". The Nagios daemon can be configured to react to these return codes, notifying administrators via email or SMS, for example. A *service check* in Nagios terms is simply comparison of a metric value for some service (broadly defined) against a threshold. This could be done either periodically or on demand. The term "service" is quite broad. It can mean an actual service that runs on the host (POP, SMTP, HTTP, etc.) or some other type of metric associated with the host (response to a ping, number of logged in users, free disk space, etc.).

3.3 Current Mobile Monitoring Solutions

To motivate the need for our approach, we discuss the current mobile monitoring solutions and their ability to monitor large-scale commercial data centers. There exists a wide variety of monitoring tools, both commercial and open source, e.g., Nagios, Icinga, Ganglia, and Zabbix. All these systems have their particular strengths in monitoring the machines but the interface with the mobile device poses challenges. From an architectural point of view, all currently used solution are similar and operate as shown in Figure 3.

The baseline for consuming events from these systems is the traditional approach of sending email or text messages to the mobile device. Such a solution, however, is not able to provide system administrators with an interactive access channel to gain richer insights of the current situation in the data center.

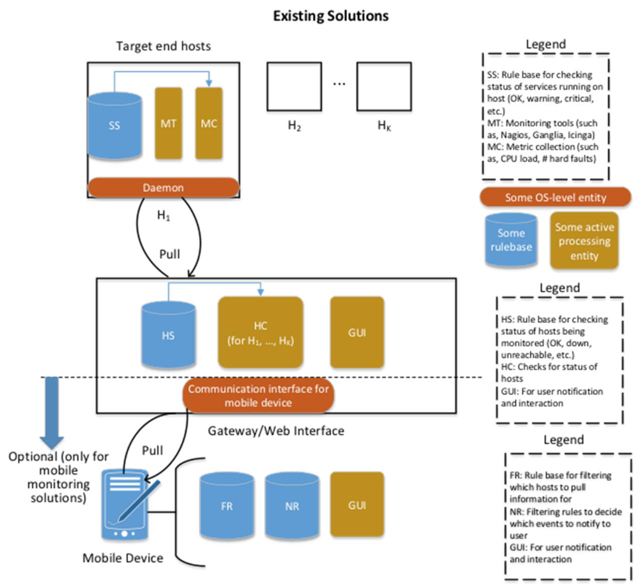

Existing Solutions
Target end hosts
SS
MT
MC
Daemon
H$_2$
...
H$_K$
Legend
SS: Rule base for checking status of services running on host (OK, warning, critical, etc.)
MT: Monitoring tools (such as, Nagios, Ganglia, Icinga)
MC: Metric collection (such as, CPU load, # hard faults)
Some OS-level entity
Some rulebase
Some active processing entity
H$_1$
Pull
HS
HC (for H$_1$, ..., H$_K$)
GUI
Legend
HS: Rule base for checking status of hosts being monitored (OK, down, unreachable, etc.)
HC: Checks for status of hosts
GUI: For user notification and interaction
Communication interface for mobile device
Gateway/Web Interface
Optional (only for mobile monitoring solutions)
Pull
Mobile Device
FR
NR
GUI
Legend
FR: Rule base for filtering which hosts to pull information for
NR: Filtering rules to decide which events to notify to user
GUI: For user notification and interaction

Figure 3: A high-level schematic showing how system monitoring is done today. An optional part shows the architecture when the system administrator is using a mobile device for her tasks.

Beyond that, there exist a plethora of mobile applications that are able to communicate with monitoring software through a web interface to receive alerts and notify the system administrator. The most popular and widely used applications either utilize Nagios or Zabbix to receive alerts.

3.3.1 *Mobile Systems Managament Applications*

There exist a plethora of mobile applications which enable users to monitor data center activities using their mobile device. Two of the most popular such applications are aNag [3, 4] and Zax Zabbix Systems Monitoring [10, 16]. The entire class of applications is able to provide functionalities like: communicate with the Nagios web interface to receive alerts, communicate with multiple Nagios web interfaces, allow users to choose polling interval for retreiving alerts (1 minute - 2 hours), provide quiet hours where users will not receive alerts, etc.

However, these applications do not allow system administrators to define custom user-defined threshold alerts. In order to change the threshold values for alerts, it is required that the individual configuration files on the target machines are modified. In the scenario where different system administrators would like to receive an alert for a given service, but at different thresholds, they would be required to setup additional service checks on configuration files. As a result, this prevents administrators with overlapping, but non-identical interests from monitoring the same service alert data without creating additional service checks on the target machine. When an excess number of service checks are executed on the target machine, we observe that a percentage of the alerts get dropped.

In current solutions, either setting up the rule matchings initially or making a subsequent change involves change of configurations on each target machine within the data center. In a commercial data center it will become tedious to manage independent configuration files for the various target machines. Further, existing solutions are not able to provide the context around the failure in an intelligent manner. More specifically, it can be hard coded to provide a certain amount of data around the time of the failure. However, the precise demand for context is rarely fully known in advance of an event and might be an individual choice of each system administrator and also depend on the event of interest that matched.

3.3.2 *Comparative system: aNag*

For an in-depth analysis of the shortcoming of current solutions, we consider a representative application called aNag which allows system administrators to monitor Nagios and Icinga alerts using the mobile device. aNag is the most recommended mobile application by Nagios for monitoring alerts. It also has the highest rating on Google Play (rating of 4.8/5) among all mobile monitoring apps. The application communicates directly with the Nagios/Icinga web interface to retrieve alert information. aNag allows users to choose how often the application should communicate with the web interface to receive alert information, going up from 1 minute to 1 hour. The application allows user-specified notifications on the mobile device, e.g., showing an alert only when some service on any machine goes to the state *CRITICAL* but not for *WARNING*. Other supported filters include time of day notification (enforcing *silent time periods*) and automatic background refresh. Conceptually, this and other mobile monitoring solutions are pull-based with a given periodicity in time. They are not event-based in which events get pushed to the mobile device only when they match the user interest profile.

Not surprisingly, the application uses a large amount of bandwidth when monitoring target machines. On an average, monitoring a single host machine causes the mobile device to use approximately 15 KB per minute. However, when trying to manage the activities of a commercial data center, the number of target end machines to be monitored is large—a middle sized data center will have 2,000 to 5,000 machines—and the bandwidth consumption gets multiplied by that. This happens because the application does not have any intermediate entity to find commonalities across machines and only push the relevant information. In summary, the bandwidth usage of mobile monitoring solutions is a real existing problem.

4. MAVIS: DESIGN AND IMPLEMENTATION

In this section, we will describe the design and implementation of our target application MAVIS for system management. We use the MEDAL middleware framework as described in Section 2 to create the system management application, MAVIS.

In the MAVIS application, the biggest conceptual change is the interposition of an intermediate server between the mobile client application and the monitoring software on the target machines. This intermediate server takes off the intensive computations from the mobile devices and provides a rich set of analytics functionalities leading (in some cases) to the generation of alerts, which are pushed to the mobile application. The mobile client, is able to receive the alerts and visualize the log of alert data, thus providing the necessary context for the alert, which is useful to the system administrators to debug the problem. Figure 4 represents the various components of the application and the interactions between them.

4.1 Format of subscriptions

It is essential that users can create on-the-fly subscriptions instead of modifying independent configuration files on the target end hosts. This relieves the user from performing the tedious task of modifying independent configuration files on each of the target end hosts within the data center. This also could result in an overhead on the target machine if the user tries to create multiple service checks for the same service.

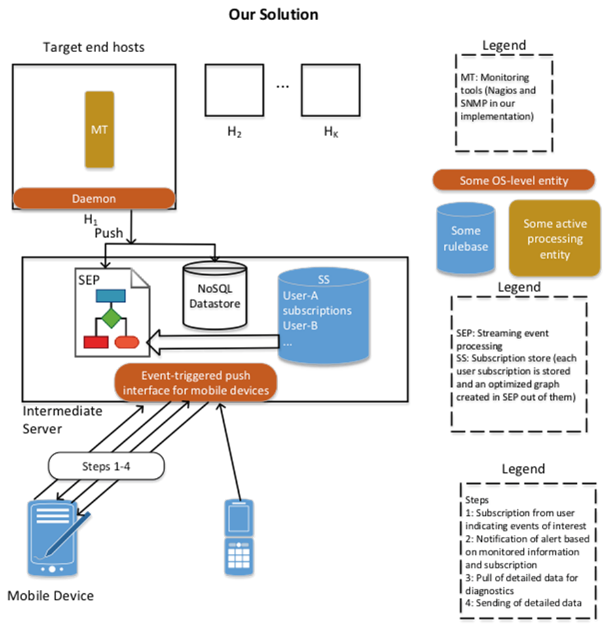
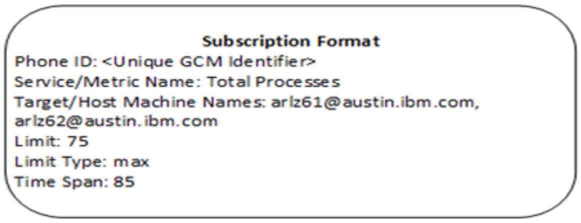

Our Solution
Target end hosts
MT
Daemon
H1
H2
HK
Push
SEP
NoSQL Datastore
SS
User-A subscriptions
User-B
Event-triggered push interface for mobile devices
Intermediate Server
Steps 1-4
Mobile Device
Legend
MT: Monitoring tools (Nagios and SNMP in our implementation)
Some OS-level entity
Some rulebase
Some active processing entity
Legend
SEP: Streaming event processing
SS: Subscription store (each user subscription is stored and an optimized graph created in SEP out of them)
Legend
Steps
1: Subscription from user indicating events of interest
2: Notification of alert based on monitored information and subscription
3: Pull of detailed data for diagnostics
4: Sending of detailed data

Figure 4: A high-level schematic of our system management application called MAVIS. The key characteristic is that the intermediate server processes streams of data from all the monitored target end hosts and delivers focused events to the resource-constrained mobile devices. At the same time, it preserves the ability to go back and extract contextual information around the event of interest.



Subscription Format
Phone ID: <Unique GCM Identifier>
Service/Metric Name: Total Processes
Target/Host Machine Names: arlz61@austin.ibm.com, arlz62@austin.ibm.com
Limit: 75
Limit Type: max
Time Span: 85

Figure 5: MAVIS Subscription Format

The format of a subscription that is created by MAVIS is provided in Figure 5. The application is able to create a subscription for any service which is monitored on the specified target machine. On creating a subscription, the user is registered to receive an alert when the subscription condition is met. The triggering of an alert is dependent on the "Limit" and "Limit Type" parameters of the specific subscription. The "TimeSpan" parameter within a subscription is used to define how much prior raw alert data should be pulled from the target host. The subscription in Figure 5 triggers the initial alert when the Total Processes on the specified target machines exceeds 75. This example shows the spatial as well as the temporal component in our subscription language. The spatial component shows up through the combining of metric values from two different machines. The temporal component shows up through the use of metric values over a window of time.

The amount of prior data pulled from the server depends on the "Time Span" parameter when creating the initial subscription. When making the initial pull request, the application gets the number of minutes specified in the Time Span parameter of the specific subscription. If we assume that Time Span is x minutes, then the

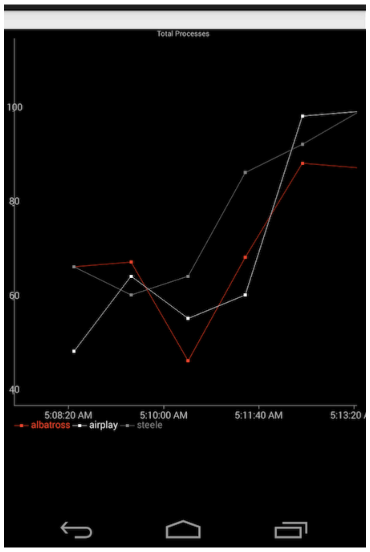


Figure 6: Screenshot of alert data log within mobile application

application can pull at most the past x minutes of alert data. After the initial pull request, the application periodically pulls alert data every minute to get the new alert value. Once the pull request gets an alert value that will not result in an alert trigger, it stops pulling alert data. We provide an interface to visualize this data on a graph, with the metric values on the y-axis and time on the x-axis as shown in Figure 6. The graph of alert data is an essential element in helping the human identify the probable cause of the alert failure by correlating the alert data for the various monitored metrics. We do not make any attempt to automate this debugging process in our current system.

4.2 Intermediate Server

Our solution is agnostic to the monitoring software. We require a Streams converter to be written to convert the stream of monitored information into a form that Streams can ingest. For our implementation and evaluation, we use Nagios and SNMP running on Blade Servers and their respective Management Module.

The goal of the intermediate server in our design is to act as an efficient bridge between the numerous servers that are to be monitored and the mobile clients used by multiple system administrators in the field. It is used for gathering data from all target servers that are to be monitored, matching alerts defined by an end user, and pushing out mobile alerts to the system administrators in the field.

This machine is kept in close network proximity with the data center to be monitored, and will be the primary machine tasked with monitoring communication with the target servers. Requests for information previously sent from administrator's desktop or mobile devices directly to target servers, are sent instead to this intermediate server. In addition to the most recent monitored data being stored on the intermediate server, it is imperative that a suitable amount of historical data be saved on the machine for some period of time.

Because all monitored data is now being funneled to the intermediate server, user-defined alerts can be matched against live incoming data. These user-defined alerts, or subscriptions, must also be stored on the intermediate server along with some identification pairing certain alerts with the interested end user's mobile device. The final piece of the intermediate server is the ability to push out

Readings from Management Module (SNMP)	**Nagios Collected Data**
Front Panel Ambient Temperature	Percent of Disk Space Free
Management Module Temperature	MBs Free on Disk
BladeCenter Blower Status (Error/OK)	Number of Users on Blade
BladeCenter Blower Speed (Percentage)	Total number of Processes
Management Module Voltage Readings	CPU Core Temp readings
Bladecenter Power Status (Off/On)	CPU load average per core over last 1,5,15 minutes
Generic Health Status (Warning/OK)	Blade Memory Free/In Use (MB)
Blade Server Voltages	

Table 1: List of metrics monitored in MAVIS.

timely alerts to mobile devices, giving users the ability to request more detailed information if desired.

The intermediate server communicates regularly with the target end hosts, which are situated within data centers. In order for the intermediate server to communicate and retrieve monitoring information from these target hosts, they must be running a monitoring tool such as Nagios, which can execute services and collect the monitored information and place it in a central location. The intermediate server can communicate with any server that uses such monitoring tools to collect alert data. The intermediate server will periodically communicate with the target machine and request it to provide alert values for the monitored services. Once the alert values are obtained, it will pull the monitored data from the target host.

The Intermediate server uses the pulled monitor data to gather information on the status of both the individual Blade Servers, as well Blade Centers in which they are housed. Metrics monitored using Nagios are pulled from the individual target servers. The Management Module provides additional information for a set of Blade Servers in a single Blade Center. A full list of readings monitored in our current implementation can be found in Table 1. The Management Modules in the server center use SNMP in monitoring the status of the entire Blade Center to which it belongs. Pulling of this information is accomplished through SNMP get commands.

4.3 Communication to the mobile application

The intermediate server communicates with the MAVIS mobile application using the Google Cloud Messaging (GCM) push feature. When the 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. The GCM framework guarantees that it will provide a unique identifier for each smartphone that is registered to receive push notifications from the service. We assume that each system administrator will have their own device for receiving alert notifications. So, we send the GCM identifier to the intermediate server, so that it can uniquely identify the mobile device when it needs to send an alert to the device. GCM also comes with the functionality that an offline device can have the alert buffered at a Google server and delivered to it when it comes online.

5. EVALUATION

In this section, we present the evaluation of MAVIS. We first lay the basis for the rules we use in our evaluation by doing a survey of possible rules for checking data center failures. The survey is done among a subset of system administrators at Purdue's IT organization and IBM Research. Our evaluation is structured around four experiments.

1. What is the resource cost of supporting multiple system administrators monitoring the same set of machines (but different aspects of each machine) using an existing solution? We use the Nagios monitoring solution for this evaluation due to its acceptance as the best-of-breed monitoring solution.

2. What is the detection latency for MAVIS when there are failures? We measure the processing time at the intermediate server and the total time for a user at a mobile device to receive an alert.

3. What is the resource utilization of MAVIS and aNag as a function of the number of machines being monitored, when there is no failure occurring? We measure the resource utilization at MAVIS since in the absence of failure, there is little that is happening at the mobile device. For aNag, we measure the resources spent at the mobile device since that is the only entity (apart from the target end host) that is involved. From the experimental results, we can extrapolate to how many target end hosts each of MAVIS and aNag can handle.

4. For the use case with Purdue's production clusters, we queried the system administrators for the most troublesome problems and set up and ran MAVIS with the subscriptions corresponding to them. Here we quantify the resource cost of such an execution and extrapolate to how many target end hosts MAVIS can handle.

5.1 Evaluation setup

For the scalability (with number of machines) experiment, we use monitored target host machines within ITaP, Purdue's central IT organization. ITaP monitored data is available for approximately 3,000 different target machines. Each target machine has between 6 and 12 services being monitored on them. System Administrators tend to reduce the number of service checks on the machines in order to avoid an excess load on them. This is a common setup for a mid-sized data center.

We monitor the ITaP target machines using Nagios in two different ways. These are two widely used alternative deployment styles.

1. Each target machine has its own instance of Nagios running and has its own web interface which keeps track of all the monitored information. In the case of MAVIS, the intermediate server will pull the monitored information from each individual host, while in the case of aNag, the mobile software will do this.

2. We use a central server, called a *gateway*, which collects all the monitoring information from each of the target machines. The central server has a web interface which maintains all the monitoring information for each target host machine. In the case of MAVIS, the intermediate server will be pull the monitored information from the gateway, while in the case of aNag, the mobile software will do this.

To inform the design of our system monitoring application, we conducted a survey of system administrators in Purdue's central Information Technology organization (called ITaP) and IBM Research Information Systems group. 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 13 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 six 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)
(iv) Alert (within a few seconds) if the management module on the Blade chassis becomes unreachable. (Instantaneous rule)
(v) Average server temperature for the past N minutes is above X degrees. (Temporal rule)
(vi) Average disk space free for the past N minutes is less than X per cent. (Temporal rule)

We observe that these top six conditions are split evenly 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. We believe that our MEDAL middleware provides an infrastructure that fulfills this requirement.

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.

5.2 Experiment 1: Supporting multiple system administrators with existing solution

We asked the question how would an existing monitoring solution be adapted to serve the use case of multiple system administrators having overlapping interests about machines in a data center. For a representative example of an existing monitoring solution, we used Nagios. An intuitive way in which one may think of supporting the use case with Nagios is that multiple service checks are performed on a target end host, sometime for the same metric corresponding to different trigger thresholds for different system administrators. For example, consider that one system administrator is in charge of the cooling subsystem and is interested in large changes in temperature profile (say, temperature exceeds 90 degrees continually for one hour), while another system administrator is looking at more fine-grained changes in temperature brought about by unequal distribution of computation load (say, temperature fluctuates by 5 degrees or more in a 5 minute interval on one rack). With Nagios, there will be separate web interfaces for each system administrator and each service check will run completely independently

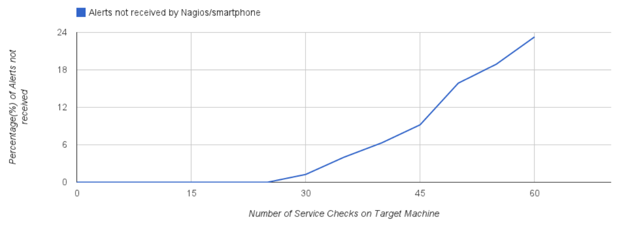


Figure 7: Percentage of alerts not received by baseline Nagios and aNag as a function of monitored target machines

of each other, even when two service checks are based on the same metric. A second drawback, though unrelated to this experiment, is that Nagios does not provide a way of doing aggregation (temporal in the above example) and basing alerts on the aggregated value. Thus, going back to the way in which Nagios can be made to meet the use case, the number of service checks will increase with the number of system administrators interested in some aspect of the machine. Intuitively, this would put pressure on the target end host, which is particularly undesirable since these hosts are the workhorses of a datacenter and the datacenter operator wants to run as little as possible on these beyond the core application. We quantified the effect on the target end hosts through this experiment.

When using a central server to monitor services on the target machines, while executing more than 30 service checks on each target machine we notice that a fraction of the alerts get dropped. Figure 7 shows the variation in the percentage of alerts that do not get received by the mobile device. At 60 service checks for each machine, more than one in 5 alerts is getting dropped. Though we mentioned that it is typical to only have 6 to 12 services monitored per target machine, with just 3-5 system administrators being interested in the machine, the number would exceed 30. The number of service checks can also increase if the administrator creates spatial or temporal checks by manually writing the code for it. Thus, for example, she may have a rule for fine-grained averaging, say every 5 minutes, and a coarse-grained averaging, say every 1 hour. This problem of dropped alerts arises because the target machines are unable to provide the Nagios server the alert information for all the services within one minute. We had set one minute as the interval with which the Nagios server pulls the status of the service checks from each target end host. This time period of pulling (one minute) is quite a typical deployment since the alerts are time critical events. Hence, the central Nagios server does not receive all the alerts, which is a particularly worrisome occurrence for a system administrator interested in knowing about the well-being of the machine in a timely manner. It is not the volume of data that is exchanged, but rather the hand-shaking overhead between the Nagios server and the target end hosts that creates this bottleneck.

When monitoring target machines using MAVIS, we do not have this problem because each metric is monitored only once on a target end host, irrespective of the number of system administrators, and sent to the intermediate server for further processing and alert generation. Thus, the metric collection and the monitoring rule matching are separated in the MAVIS intermediate server. Reassuringly, and not surprisingly, for all the experiments with MAVIS, we have observed no dropped alert.

5.3 Experiment 2: Latency of error detection in MAVIS Intermediate Server

It is essential that the MAVIS middleware is able to efficiently identify alerts and disseminate the data to the mobile devices. In

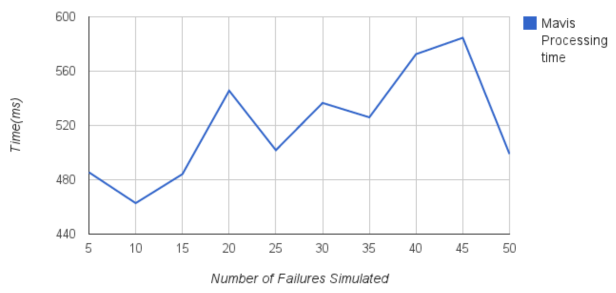


Figure 8: Average alert trigger processing time by MAVIS as a function of the number of failures per minute, cumulative across all the target end hosts being monitored

this experiment we measure the time taken by the MAVIS middleware to detect a failure and trigger an alert. In order to measure the perfomance of MAVIS, we use it to monitor four IBM Blade Servers and simulate failures within the Blade Servers. We simulate these failures by changing the threshold value for alert triggers within the MAVIS middleware.

In evaluating the response time of our solution, scaling up to a ceiling of 50 alerts triggered per minute had a negligible effect on the time the MAVIS middleware takes to process and generate alerts as seen in Figure 8. The number of alerts is cumulative for all the target end hosts that are beig monitored. We choose 50 alert triggers per minute as a reasonable ceiling, because typical users subscribe for alerts that are rarely triggered. If they were to subscribe for alerts that trigger very often, they would start disregarding the alerts and would not use them for statistical analysis. Due to the nature of GCM, we cannot guarantee the delivery time for alerts once the MAVIS middleware invokes the GCM Push service. However, during our evaluation we obseved that the time taken to receive alerts on the mobile device varied between 324 and 1638 milliseconds. The time is end-to-end, being measured from when a service check is performed on a Blade Server to when the notification is visible on the mobile device.

5.4 Experiment 3: Scalability of monitoring with MAVIS vis-à-vis aNag

As a resource rich server component, the MAVIS middleware does not need to worry about utilizing excessive bandwidth, or consuming excessive power. We measure the resource utilization of the middleware through it's CPU and Memory utilization by monitoring the target machines located within ITaP. We measure them as a function of the number of service checks being monitored, rather than the number of target end hosts because the hosts have a variable number of service checks.

Figure 9 shows the CPU usage of MAVIS as the sum of utilization of all the related processes running on a single core. We used a Dell PowerEdge server with 12 CPU Cores. The result is a roughly linear increase in CPU utilization which will likely reach a limit at roughly 100,000 service checks in our current solution, when the most intensive process, our MongoDB storage solution, will create the bottleneck. Passing this limit would simply require a distributed setup of the storage element of our solution, which MongoDB supports.

Figure 10 shows the memory utilization on MAVIS as we increase the number of service checks on the target machines in the data center. We observe a linear increase in Memory utilization

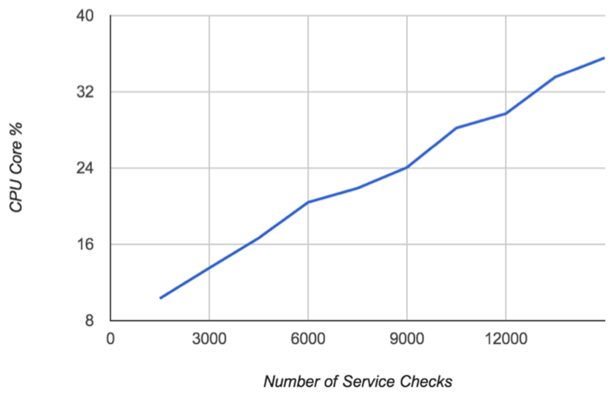


Figure 9: CPU utilization by MAVIS as a function of number of service checks

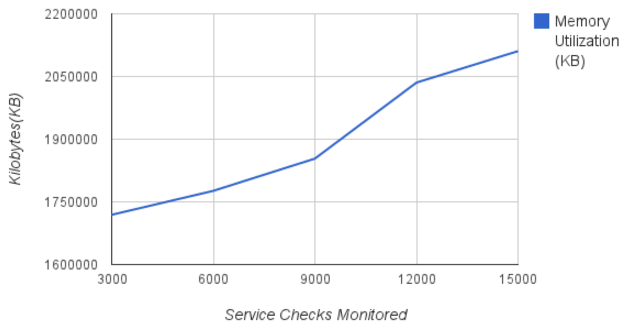


Figure 10: Memory utilization by MAVIS as a function of number of service checks

which will reach a limit when we monitor more than 100,000 service checks. Since MAVIS eliminates the need to duplicate service checks for different thresholds, each target machine will only have 1 service check per unique metric. Hence, 100,000 service checks is a reasonable ceiling.

In order to measure the scalability of the state-of-the-art mobile monitoring solution aNag, we monitor up to 130 target machines within ITaP using the application. We use a monitoring frequency of aNag of once every minute for each target end host. We measure the resource utilization by aNag in terms of the bandwidth utilization and the number of threads.

5.4.1 *Bandwidth Utilization*

We monitored up to 130 different target machines using aNag where each target machine was monitoring anywhere from 6 to 12 services. Figure 11 shows the bandwidth utilization of aNag as a function of the number of service checks. Using [11] as a reference for how bandwidth hungry popular applications are, we find that 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 hungries bandwidth usage limit (115 MB/hour for the hungriest of these 50 applications) while monitoring 130 target machines. The number of target machines being monitored in these cases is only a small fraction of the total number of target machines present in middle and large-sized data centers. In both

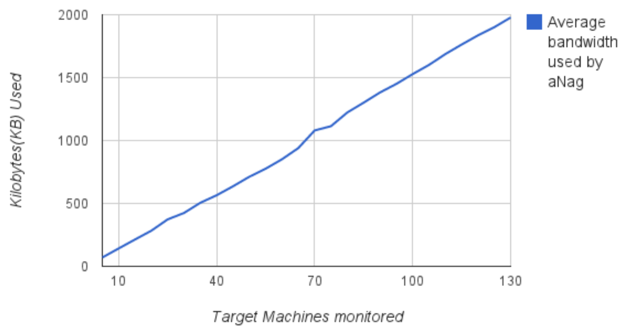


Figure 11: Bandwidth utilization by aNag as a function of monitored target machines

Nagios setups (referring to the two setups described in the beginning of this section) the same amount of monitored data is being brought into the mobile device. As a result, we observe similar bandwidth usage trends in both Nagios setups. MAVIS avoids the excessive bandwidth usage by choosing to have the mobile device receive alerts. MAVIS only utilizes bandwidth when a push notification is sent to the mobile device. Furthermore, we do not send all of the prior alert data within a push notification, and allow the user to decide when to obtain the log of prior alert data. Hence, the only time MAVIS uses bandwidth is during an alert trigger, and the typical user will generally subscribe for alerts that rarely get triggered.

5.4.2 *Thread Utilization*

In order to measure the scalabilty of aNag we monitored up to 100 target machines, where each target machine had between 6 and 12 Nagios service checks. Figure 12 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, the first of the two modes of Nagios deployment mentioned at the beginning of this section. As a result, aNag must create threads to establish a network connection with each target machine. We find from Figure 12 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.

The MAVIS systems management application does not experience this drawback, since the mobile application waits for the middleware to send alerts and does not create threads per target end host. The application consistently uses between 657 and 660 threads irrespective of the number of monitored target machines.

5.5 Experiment 4: Use case with real system administration problems

In discussion with system administrators from ITaP, we were made aware of 2 failures that have not previously been caught in their current Nagios-based monitoring solution. The first problem, involved a failure where a cluster was made unusable due to increasing temperature. This failure occurred as a result of a failure in the HVAC [8] monitoring system. Through the intermediate server, MAVIS provides the ability to monitor the variation in temperature

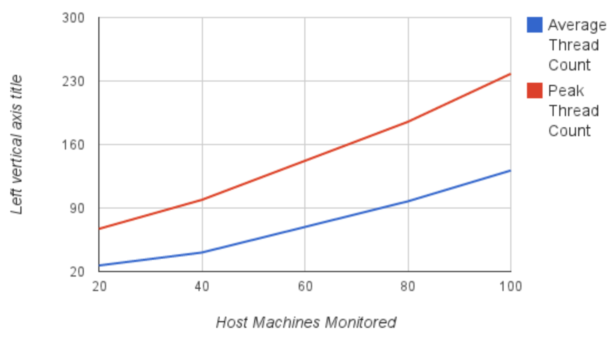


Figure 12: Average and peak thread utilization by aNag as a function of monitored target machines

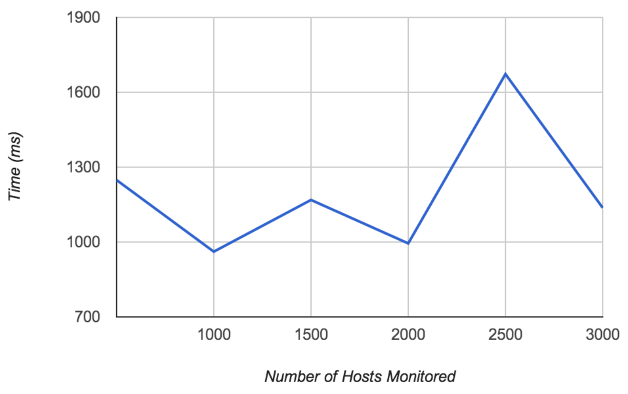


Figure 13: Processing Time by MAVIS for monitoring the two most vexing ITaP data center problems as a function of the number of target end hosts being monitored

on the entire data center by checking the temperature through the target machine sensors.

Another challenging problem that ITaP encountered is predicting DIMM errors. To catch these errors, we use the intuition that looking for recoverable memory errors is a good indication of an unrecoverable memory error coming down the line. According to the most extensive study of DRAM failures in the wild [15], an uncorrectable error is preceded by a correctable error 70-80 percent of the time. A correctable error increases the probability of an uncorrectable error by factors of 9-400. By using our solution to persistently check each machine's BMC System Event Log using IPMI we are able to monitor the rate of correctible ECC errors on each target. To provide an alert when the probability of a future uncorrectable DIMM error is high, we create a subscription that will send an alert to a user's mobile device when the total number of errors corrected by ECC in the past 24 hours exceeds a given threshold.

Figure 13 shows the average processing time for the MAVIS framework as a function of the monitored target machines. The target machines run scripts to identify the temperature and DIMM alert values. We observe that for these experiments the amount of data passed through Infosphere Streams is well below the processing capability of Streams, and hence we observe an approximately

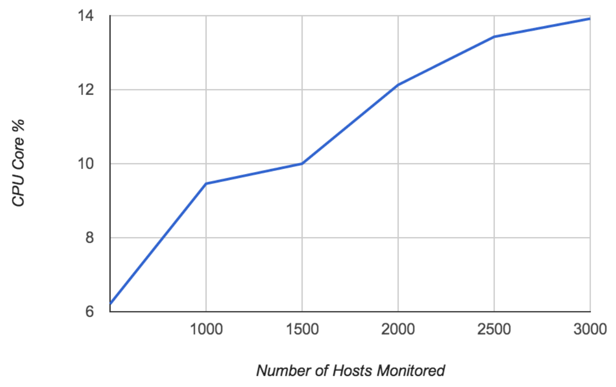


Figure 14: CPU utilization by MAVIS, for monitoring the two most vexing ITaP data center problems as a function of the number of target end hosts being monitored

horizontal trend in the average processing time. Figure 14 shows the average CPU utilization by the MAVIS application. As shown in the previous experiment, we observe a linear increase in the CPU utilization. Extrapolating the CPU consumption, MAVIS should be able to support 50,000 target end hosts, monitoring these two service checks corresponding to the vexing problems faced by ITaP in its data centers. However, it is likely that the number of network connections will force a lower cap on the number of target end hosts. We discuss in the "Discussion" section, how the pressure on the number of network connections can be reduced.

6. DISCUSSION

In the design of our solution, the MEDAL middleware periodically establishes a connection with each of the target host machines. As the number of target machines within the data center increases, it stresses the scalability of the middleware since the operating system tends to limit the number of concurrent network connections. Such a scenario can eventually prevent MEDAL from communicating with each of the target machines periodically. This problem can be contained by multiplexing multiple IP connections through a single network socket. Furthermore, we can setup a distributed environment with multiple MEDAL middleware servers, such that each server is responsible for monitoring a subset of the overall target machines. Our implementation of MEDAL using IBM Streams makes it relatively simple to set up this distributed monitoring environment.

The problem of system administration from mobile phones has clearly motivated our work on MEDAL and set some of the requirements for the system. However, the middleware is by no means restricted to this application, but a variety of unrelated applications can equally benefit from our design if access through the mobile phone is desired. For instance, many applications around Smart Home and the Internet of Things have similar requirements in terms of being able to subscribe to alerts for complex events and further drill down into the data that triggered the alert. The higher the degree of automation becomes, the more sensor data is going to be processed so that at some point full warehousing might not be feasible anymore. Further, the focused information delivered to the human, together with supporting context information, may be suited to human cognitive capabilities and ease the task of taking a response action.

Privacy could be an orthogonal reason to avoid warehousing. If the subscriber for the event is an external entity, it might get permission to subscribe to, e.g., error or maintenance events caused by a set of devices and for only in this case would be permitted to access the data related to the generated alert whereas the remaining data would not be exposed.

7. RELATED WORK

Conventional monitoring of systems and networks.

The science of monitoring computer systems and networks for the purpose of early detecton of problems is quite mature. However, this area continues to see improvements driven by a multitude of factors — larger scale of systems, newer kinds of workloads, newer computer architectures (such as, the emergence of GPUs, FPGAs, and other accelerators), and newer failures and attack modes. In this realm, open source monitoring softwares seem to hold sway. The most popular ones are Nagios (which forms the base for our system), Ganglia [6], and OpenNMS [14]. A hallmark of these tools is that they are flexible and configurable. The underlying intution, and sound intuition at that, is that the system developers cannot foresee the diversity of computer systems and networks that they will have to monitor and therefore they do not overarchitect their system to provide support for all variety of computer systems and networks. The tools monitor the status of network devices and their services and notify the system administrators when problems have been found. The core of the Nagios engine, for example, is a scheduler daemon that regularly probes specified network devices and their services. When problems occur Nagios alerts network administrators through notification channels such as email and instant messages service. Administrators can open the web interface to browse for the status information, event logs, and reports from anywhere via the internet.

Mobile monitoring solutions.

We have seen a slow but steady increase in the number of mobile monitoring solutions for the two dominant mobile platforms—Android and iOS. We have selected aNag as our baseline monitoring solution to learn from and compare MEDAL to. Many of the mobile applications are clients of Nagios (including aNag). Some other popular mobile clients (listed at [13]) are iNag, Mobile Admin, and TeenyNagios. They all have pull functionality from the smartphone to the Nagios datastore. Some are web clients (TeenyNagios) and some are native applications (aNag and iNag). None of them perform the aggregation and alert generation functionality as MEDAL' intermediate server does. While there is a slew of mobile monitoring solutions — basically for each possible combination of mobile device platform and monitoring software (Nagios, Icinga, etc.) — they all share the same architecture as aNag, i.e., mobile device pulling information from the target end hosts.

Closely related solution: Zabbix.

The existing mobile monitoring solution that appears closest to MEDAL is the Zabbix enterprise monitoring system and its mobile frontend called Zax [10]. Zabbix is a fairly lightweight monitoring application that is able to manage thousands of items with a single-server installation. A Zabbix agent is deployed on a monitoring target to actively monitor local resources and applications (hard drives, memory, processor statistics etc). The agent gathers operational information locally and reports data to Zabbix server for further processing. In case of failures, the Zabbix server can actively alert the administrators of the particular machine that reported the failure. A Zabbix proxy is another member of the Zabbix suite of programs that sits between a full blown Zabbix server and a host-oriented Zabbix agent. It is used to collect data from any number of items on any number of hosts, and it can retain that data for an arbitrary period of time, relying on a dedicated database to do so. It also limits itself to data collections without triggering evaluations or actions.

While similar in architecture to MEDAL, Zabbix has several drawbacks vis-à-vis MEDAL. 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 easily 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 MEDAL, 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.

Another seemingly close solution is called PCMonitor [12]. It has the architecture of three tiers — the target end hosts, the intermediate server, and the mobile clients. However, it is closed source and has little configurability, say with respect to how aggregation is done of alerts across multiple servers, how changing interests may be accommodated, etc.

8. CONCLUSION

The ubiquitous availability of mobile phones and widespread network connectivity have led to a demand for real-time information about the state of computing devices and infrastructure such as data centers. One example where this is highlighted is in system management in large commercial data centers. System administrators would like access to richer data set from the mobile device while minimizing the data monitoring overhead on data center host machines, timely processing of monitored data, minimizing the resource consumption on the mobile devices, and meeting the dynamically changing needs of system administrators.

These requirements motivated our design of MEDAL, a middleware platform for evaluating high-volume data streams, generating alerts based on dynamic rule-based subscriptions, retaining sufficient data around detected events for effective problem analysis, and efficiently disseminating this content to subscribed mobile devices. Our evaluation shows that mobile system management solutions can be much more efficiently implemented using our MEDAL middleware and the scalability and resource utilization on the mobile devices are significantly better than existing state-of-the-art mobile systems management applications like aNag. For example, while aNag runs into resource exhaustion at less than 600 hosts, while extrapolating MEDAL's performance implies it will be able to handle 100,000 machines.

9. REFERENCES

[1] S. Babu and J. Widom. Continuous queries over data streams. *ACM Sigmod Record*, 30(3):109–120, 2001.

[2] 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.

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

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

[5] FreeBSD. Clock Synchronization with NTP. `http://www.freebsd.org/doc/handbook/network-ntp.html`.

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

[7] Google Inc. GCM Advanced Topics. `http://developer.android.com/google/gcm/adv.html`.

[8] K. D. Hoog and N. P. Knobloch Jr. Hvac remote monitoring system, May 7 2002. US Patent 6,385,510.

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

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

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

[12] MMSOFT Design Ltd. PCMonitor. `http://mobilepcmonitor.com/`.

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

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

[15] B. Schroeder, E. Pinheiro, and W.-D. Weber. Dram errors in the wild: a large-scale field study. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 193–204. ACM, 2009.

[16] Zabbix SIA. Zabbix - The Enterprise-class Monitoring Solution for Everyone. `http://zabbix.com/`.