InteMon: Intelligent System Monitoring on Large Clusters

Evan Hoke Jimeng Sun Christos Faloutsos
Computer Science Department
Carnegie Mellon University
Pittsburgh,USA
ehoke@andrew.cmu.edu, {jimeng,christos}@cs.cmu.edu

ABSTRACT

InteMon is a prototype monitoring and mining system for large clusters. Currently, it monitors over 100 hosts of a prototype data center at CMU. It uses the SNMP protocol and it stores the monitoring data in an mySQL database. Then, it allows for visualization of the time-series data using a JSP web-based frontend interface for users.

What sets it apart from other cluster monitoring systems is its ability to automatically analyze the monitoring data in real time and alert the users for potential anomalies. It uses state of the art stream mining methods, it has a sophisticated definition of anomalies (broken correlations among input streams), and it can also pinpoint the reason of the anomaly. InteMon has a user-friendly GUI, it allows the users to perform interactive mining tasks, and it is fully operational.

1. INTRODUCTION

Stream monitoring and stream mining have attracted a lot of research interest recently. Here we showcase one of the latest stream mining algorithms, SPIRIT [9], as applied on a real setting that involves monitoring the machines of a large data center.

In recent years, large computational clusters become a popular trend in production environments. This architecture shift brought up many challenges for the designers and system administrators such as scalability, reliability, heterogeneity and manageability. Exactly because of these challenges, monitoring systems have become of great importance. They provide a way for users to remotely monitor and tune the systems.

Several monitoring systems have been developed mainly addressing the issues on scalability and reliability [10, 11, 3] (details are introduced in Section 2). And most of monitoring systems provide a graphical interface for viewing the cluster behaviors in real time and also log the raw monitoring data. However, these approaches have two big drawbacks:

First, there is no such manpower to make effective use of those real-time monitoring. In reality, the administrators usually find out the errors or failures by other means such as user complaints, then they realize that the monitoring system actually has been observing the unusual behaviors for a long time. Because of the large volume of data, it is almost impossible for human to directly utilize the monitoring information. As a result, the monitoring system becomes a not-so-useful tool for verifying error and failure in retrospection.

Second, most of the useful but hidden information has not been discovered. For example, observing the correlations among different measurements can be very helpful for the administrator to understand the system and find out potential anomalies. However, it is hard to pick up those patterns from the raw data by just eye-balling the simple plots generated from current monitoring systems.

We believe an efficient data mining module is absolutely crucial for monitoring systems. The data mining module should be able to analyze the monitoring data in real-time and report the patterns and anomalies for inspection. The mining capability is essential for turning the passive monitoring tool into an active administrator assistant. InteMon is a intelligent monitoring system that is being built to address these issues. It is a monitoring application for large-scale clusters that provides automatic mining capability over the data in addition to the monitoring functionality as other monitoring systems. In particular, it can observe the correlations over the measurements and summarize them in a succinct manner; it can pick up the (non-trivial) anomaly behaviors and identify the sources.

The contributions of our system are :

- it processes real-time monitoring streams through simple network management protocol (SNMP) over hundreds of hosts;
- it provides an automatic data mining module over the incoming streams and records the underlying correlations;
- it spots the anomalies and identifies the source causing it;
- it has a simple web-based interface for the administrators to monitor all layers of information over the hosts (from overall behavior of the entire cluster to the performance of the individual host);
- it gives database support to manage the historical data as well as the mining results.

From a data mining perspective, we studied several important practical issues throughout the development:

1. How to find the correlation over large number of streams?
2. How to deal with missing values and asynchronous measurements?
3. How to handle bursty data?
4. How to spot anomaly and do prediction?

The rest of the demo proposal gives a brief literature survey in Section 2, the architecture of our system Section 3, discussion on stream mining issues Section 4, demonstration plan in Section 5and the conclusions in Section 6.

2. RELATED WORK

There are a number of research and commercial monitoring systems, mainly focusing on the system architecture issues such as scalability and reliability.

Ganglia [10] is a hierarchical monitoring system that uses a multicast-based listen/announce protocol to monitor nodes within clusters, and it uses a tree structure to aggregate the information of multiple clusters. SuperMon [11] is another hierarchical monitoring system which uses a custom kernel module running on each cluster node. ParMon [3] is a client/server monitoring system similar to ours but without mining capabilities. There exist commercial monitoring suites such as OpenView [5], Tivoli [6], and Big Brother [2], as well as several open-source alternatives, including Nagios [8]. These systems are primarily driven by threshold-based checks. As long as the result of a query lies within a predefined range, the service is considered to be operating normally.

While our main focus is discovering anomalous behavior to aid system administrators, others have examined ways of using this data to address performance tuning or workload distribution. Magpie [1] creates a statistical model of requests moving through a distributed system to analyze system performance. Weatherman [7] examines environmental data (e.g., workload distribution, cooling configuration, and physical topology) via a neural network to predict thermal behavior. Cohen, et al. [4] have had success using performance monitoring data and Tree-Augmented Bayesian Networks to infer whether a system is meeting its service-level objective (SLO).

In terms of mining algorithm, we focus on the SPIRIT algorithm [9], which performs PCA in a streaming fashion, discovering the hidden variables among the given n input streams, and automatically determining when more or fewer hidden variables are needed.

3. SYSTEM ARCHITECTURE

In this section, we present our system design in details. First, Figure 3.1 introduces the real-time data collection process for monitoring sensor metrics in a production data center. Figure 3.2 presents the database schemas for data storage. Then Figure 3.3 shows the functionalities of the web interface. The overall system design is shown in Figure 1.

3.1 Monitoring sensor metrics

Monitoring is done via the Simple Network Management Protocol (SNMP). The reason to choose SNMP is because

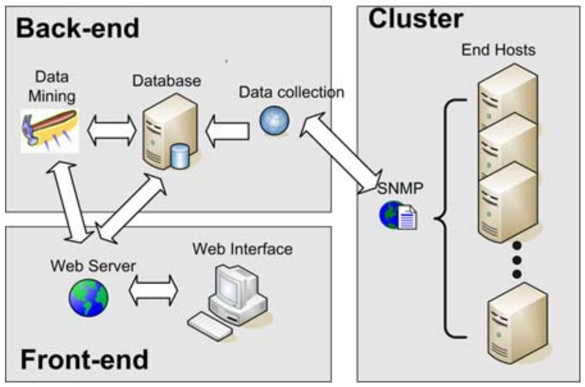


Figure 1: System design

it is a widely used protocol for managing network devices remotely, such as routers, hosts, room temperature sensors, and etc. Furthermore, some software that allows the retrieval of SNMP information can also be managed, including web servers and databases. Moreover, since SNMP has been actively running in our production systems of the data center, we decide to use this powerful infrastructure on our benefit to build this monitoring system.

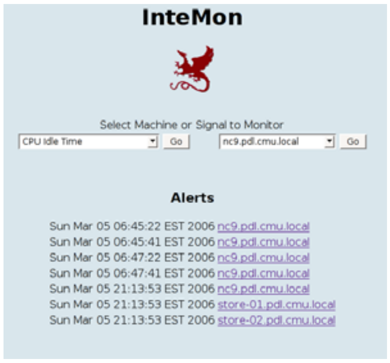
Name	Description
ifInOctets.2	Bytes Received
ifInUcastPkts.2	Unicast Packets Received
ifOutOctets.2	Bytes Sent
ifOutUcastPkts.2	Unicast Packets Sent
ssCpuRawUser.0	Unprivileged CPU Utilization
ssCpuRawSystem.0	Privileged CPU Utilization
ssCpuRawNice.0	Other CPU Utilization
ssCpuRawIdle.0	CPU Idle Time

Table 1: Examples of sensor metrics

Data collection is done through a daemon process running on a designated server. This server is configured to query a designated set of sensor metrics (see Figure 2) from all hosts in the data center through SNMP periodically. More specifically, after a fixed time period, typically a minute, the server will query, via the snmpget program, each of the hosts and store the result in a customized MYSQL database. Note that queries are spread out uniformly in the entire period to reduce the concurrent server load. And the individual client load is negligible. Our monitoring algorithm is then run across the incoming data to detect any abnormalities. The data is first grouped by signals of a given type across all hosts, then it is grouped by all signals on a given host.

3.2 Database backend

The initial data is stored in the SIGNAL_DATA table. Each row contains a value, indexed by the time, host_id and signal_id it is associated with. The monitoring table inserts but never reads from this table as the algorithm *needs to see each value only once*. The table is used primarily by the web front end for generating graphs. The tables HOST_HIDDEN and SIGNAL_HIDDEN are used to store the hidden variables found when analyzing the data grouped by host and grouped by the type of signal. These values are indexed by their time as well as their host_id or signal_id



InteMon

Select Machine or Signal to Monitor

Alerts

Sun Mar 05 06:45:22 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 06:45:41 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 06:47:22 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 06:47:41 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 21:13:53 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 21:13:53 EST 2006 store-01.pdl.cmu.local
Sun Mar 05 21:13:53 EST 2006 store-02.pdl.cmu.local

Figure 2: Data

`SIGNAL_DATA (time, host_id, signal_id, value)`
`HOST_HIDDEN (time, host_id, hidden_id, value)`
`SIGNAL_HIDDEN (time, signal_id, hidden_id, value)`
`HOST_RECON (time, host_id, signal_id, value)`
`SIGNAL_RECON (time, host_id, signal_id, value)`
`HOST_ALERT (alert_id, host_id, time)`
`SIGNAL_ALERT (alert_id, signal_id, time)`
`HOST_ALERT_WEIGHTS (alert_id, signal_id, weight)`
`SIGNAL_ALERT_WEIGHTS (alert_id, host_id, weight)`

Table 2: Database Tables

and hidden variable number. Similar to the SIGNAL_DATA table, with the same fields are the HOST_RECON and SIGNAL_RECON tables, which store the reconstructed values generated from the hidden variables found when analyzing the data grouped by host, or grouped by signal respectively.

In addition to these tables that are updated periodically there is a HOST_ALERTS and SIGNAL_ALERTS table. These tables are used primarily for generating alerts on the main page of the web front end. New rows are inserted into these tables whenever our algorithm detects an abnormality, caused by a change in the number of hidden variables. These rows contain the time of the abnormality as well as the host or signal effected and an id which is a key into either the HOST_ALERTS_WEIGHTS or SIGNAL_ALERTS_WEIGHTS tables. An entry is inserted for each host or signal corresponding to how much that host or signal contributed to the new hidden variable giving the system administrator a clue as to what may be going wrong with the systems.

The database uses round-robin table with aggregated summarization so that the data storage are fixed. Note that the hidden variables and anomalies are stored separately so that the anomaly information information is never lost.

3.3 Web interface

The web interface is JSP based and is currently running on Apache Tomcat 5.5.15 with JRE 1.5.0. The interface consists of a main page with links to monitoring page for each type of signal and for each host. Also this page lists the most recent alerts and the hosts / signals they effect as well as a list to a more extensive page of abnormalities. This gives the system administrator the pertinent information that needs to be dealt with immediately as well as to tools to investigate further. See screen shot.

The individual monitoring pages consist of three graphs(see Figure 3). These graphs are generated with the JFreeChart library version 1.0.1. Current graphs are cached for improved performance, while graphs of older data are generated on the fly. For the host monitoring page, the first graph contains a minute by minute plot of all the signals monitored for the specified host. This is similar to graphs you would see in most monitoring systems.

The second graph contains the hidden (that is, latent) variables. Intuitively if all hosts show the same pattern of CPU utilization, (eg., a daily cycle), we have only one hidden variable, which is exactly a sinusoid-like wave with 24hours period; now if half of the machines get overloaded to a 90% utilization, we need a second latent variable, constant at 90%, to capture the fact.

The last graph gives the reconstructed data. This graph uses only the hidden variables to try to approximate the original data and gives the user a feel for how well the algorithm is working. The signal monitoring pages are similar except all the signals of a given signal across hosts are plotted. On each graph, vertical bars are drawn at the locations where abnormalities occur, i.e., the number of hidden variables changes. These pages provide navigation to other monitoring pages via pull down menus as well as links to move forward and backward in time. See screen shots in Figure 3.

For each abnormality that occurs there is a link to a page providing analysis of the abnormality as shown in Figure 2. For example, if a new hidden variable is needed to describe the data, the leading signals that contribute to that hidden variable will be listed.

4. STREAM MINING

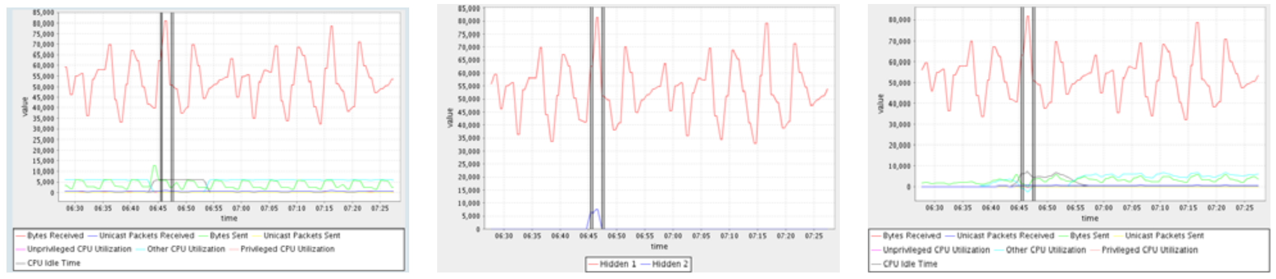
We now answer the questions listed in the end of Section 1.

Correlation Detection: Many correlation detections are available in the literature. But most of them require to do $O(n^2)$ comparisons where n is the number sensor metrics every time tick. This is clearly too expensive. We decide to use the algorithm in [9] to monitor the multiple time series, which only requires $O(nk)$ where n is the number of sensor metrics and k is the hidden variables.

For example, if we are monitoring the number of packets sent over the network across n hosts, this data is grouped into an n dimensional vector and the algorithm is applied to project this vector into a k dimensional space ($k \ll n$). The projections are the hidden variables (or the overall correlations). And the number of hidden variables are automatically determined based on the reconstruction error, i.e., given the desirable error threshold (say 10%), the algorithm will pick the smallest number of hidden variables that satisfies that.

Anomaly detection: We consider the anomalies as the sudden changes of system behavior, which is indicated by the *change on the number of hidden variables*. More specifically, if the number of dimensions in this space changes, this signifies an abnormality and its occurrence as well as the relative weights of the signals causing the abnormality. Notice that this sophisticated definition can capture anomalies way beyond values-out-of-bounds, because our system observes the past, summarizes it in a few latent/hidden variables, and issues an alert when the past is not good enough to describe the future.

Missing values: Missing values often occur in the system, usually because of SNMP goes through UDP which is unreliable but lighter weight compared to TCP. In this case, we



(a) original (b) hidden variables (c) reconstruction

Figure 3: Web interface screenshots

use the reconstruction values to substitute which is usually a good estimates. If no value is observed of a host for a long period time, that host is marked as a dead node, which is also recorded as an abnormal event.

Asynchronous arrival: The data collection is asynchronous across different hosts. However, the correlation detection algorithm require synchronous streams. Therefore, we interpolate the individual stream and correlate the streams all at the beginning of each time period.

5. DEMONSTRATION PLAN

Since the system is running in real-time on a data center, we can easily demonstrate to users how easy it is to monitor hundreds of machines using InteMon.

More specifically, the system provides user-defined monitoring functionality in addition to the existing mining process. For example, the user can specify a subset of hosts and signal and run mining module on them to find correlation and anomalies. Since we have the historical data stored in the database, users specify different time ranges or different resolution (per minute, per hour, per day, per week) for the mining module.

Overall, the demonstration will be interactive experience for users to exploit variety of real monitoring data. The goal is to show that network monitoring problem can be dramatically simplified and improved with the online data mining's help.

6. CONCLUSION

We developed InteMon: an intelligent monitoring system for a large cluster. InteMon operates on a real production cluster consisting of over 100 machines. In addition to the variety of monitoring functionalities and an intuitive graphical web frond-end, InteMon has an online data mining capability which analyzed the data in real-time to find the underlying trend/correlation and report anomalies. Finally, we hope with this demonstration we can illustrate the importance of data mining function in a monitoring application.

7. ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grants No. IIS-0209107 SENSOR-0329549 EF-0331657 IIS-0326322 IIS-0534205. This work is also supported in part by the Pennsylvania Infrastructure Technology Alliance (PITA), a partnership of Carnegie Mellon, Lehigh University and the Commonwealth of Pennsylvania's Department of Community and Economic Development (DCED). Additional funding was provided by donations from Intel, NTT and Hewlett-Packard. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

We are pleased to acknowledge John Strunk and Greg Ganger from parallel data lab at CMU for providing the invaluable data and discussion.

8. REFERENCES

[1] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan. Magpie: Online modelling and performance-aware systems. In *HOTOS*, pages 79–84. USENIX Association, 2003.

[2] Big brother. http://www.bb4.org.

[3] R. Buyya. PARMON: a portable and scalable monitoring system for clusters. *Software - Practice and Experience*, 30(7):723–739, 2000.

[4] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *SIGMOD*, 2003.

[5] Hp openview. http://www.managementsoftware.hp.com/index.html.

[6] Ibm tivoli. http://www.ibm.com/software/tivoli/.

[7] J. Moore, J. Chase, and P. Ranganathan. Weatherman: Automated, online, and predictive thermal mapping and management for data centers. In *International Conference on Autonomic Computing*, 2006.

[8] Nagios. http://www.nagios.org.

[9] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.

[10] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler. Wide area cluster monitoring with ganglia. In *CLUSTER*, 2003.

[11] M. J. Sottile and R. Minnich. Supermon: A high-speed cluster monitoring system. In *CLUSTER*, pages 39–46, 2002.