Intelligent System Monitoring on Large Clusters

Jimeng Sun Evan Hoke John D. Strunk Gregory R. Ganger Christos Faloutsos

Carnegie Mellon University

Abstract

Modern data centers have a large number of components that must be monitored, including servers, switches/routers, and environmental control systems. This paper describes `InteMon`, a prototype monitoring and mining system for data centers. It uses the SNMP protocol to monitor a new data center at Carnegie Mellon. It stores the monitoring data in a MySQL database, allowing visualization of the time-series data using a JSP web-based frontend interface for system administrators. What sets `InteMon` apart from other cluster monitoring systems is its ability to automatically analyze correlations in the monitoring data in real time and alert administrators of potential anomalies. It uses efficient, state of the art stream mining methods to report broken correlations among input streams. It also uses these methods to intelligently compress historical data and avoid the need for administrators to configure threshold-based monitoring bands.

1 Introduction

The increasing size and density of computational clusters and data centers pose many management challenges for system administrators. Not only is the number of systems that they must configure, monitor, and tune increasing, but the interactions between systems are growing as well. System administrators must constantly monitor the performance, availability, and reliability of their infrastructure to ensure they are providing appropriate levels of service to their users.

Modern data centers are awash in monitoring data. Nearly every application, host, and network device exports statistics that could (and should) be monitored. Additionally, many of the infrastructure components such as UPSes, power distribution units, and computer room air conditioners (CRACs) provide data about the status of the computing environment. Being able to monitor and respond to abnormal conditions is critical for maintaining a high availability installation.

Administrators have long relied upon monitoring software to analyze the current state of networks, hosts, and applications. These software packages continue to evolve and improve in their scalability as well as the breadth of devices and conditions they monitor. Unfortunately, with the scale of today's systems, it is still very difficult to effectively monitor an entire data center. Our target is the Data Center Observatory, a data center environment under construction at Carnegie Mellon designed to bring together automation research and real computation and storage needs.

Traditional monitoring software has three significant weaknesses that make the capture and analysis of monitoring data difficult.

Configuration: Monitoring software requires significant time and expertise to properly configure. For each data stream that the administrator intends to monitor, he must decide upon proper thresholds for the data values. That is, he must define, for each data stream, what constitutes "normal" behavior. While classes of devices or instances of applications may share a common set of these thresholds, the administrator is still left with quite a challenge. All this effort means the administrator is unlikely to take advantage of much of the information available.

Reasoning: When troubleshooting problems within the data center, simple monitoring of independent data streams is not very helpful for tracking down problems. For example, the administrator may receive an alert that an application's response time is too large, but the administrator is still left with the difficult task of determining the root cause.

Historical data: When troubleshooting, it is very useful to know how a system has performed in the past. Current monitoring software attempts to answer this by providing historical averages as a way of summarizing past system behavior. While maintaining high resolution data from thousands of data streams over a long period of time is impractical in many situations, better techniques for summarizing the data are necessary. An administrator needs to not only know averages, but variations and extremes to efficiently troubleshoot problems.

Using stream-based data mining, `InteMon` is designed to address these weaknesses of current monitoring software. `InteMon` uses the SPIRIT [17] stream mining algorithm to analyze the many data streams available in modern data centers.

`InteMon` is designed to be a monitoring application for

Proceedings of the 3rd International Workshop on Data Management for Sensor Networks (DMSN'06), Seoul, South Korea, 2006

large-scale clusters and data centers. It will complement existing solutions by providing automatic mining as well as efficient storage for the many data streams common in today's clusters. In particular, it can observe the correlations across data streams, summarizing them in a succinct manner; it can pick up anomalous behaviors that manifest as broken correlations; it can summarize historical data as compact "hidden variables" that can be used to approximately reconstruct the historical data when needed.

`InteMon` seeks to decrease the burden of system monitoring in several ways. First, it decreases the level of expertise necessary to configure the monitoring system. It accomplishes this by removing the need for the administrator to set "alert" thresholds for the incoming data. Through stream mining techniques, it learns correlations in the data stream and can flag deviations.

Second, instead of just examining each data stream in isolation, `InteMon` looks for correlations across data streams. An alert is generated when the SPIRIT algorithm detects a change in the level of correlation across data streams. The existence (or disappearance) of these correlations provides the system administrator with a starting point for troubleshooting activities.

Third, by performing a variant of incremental principal component analysis, SPIRIT [17] is able to incrementally and compactly express the correlations and variations of the data across streams as well as detect abnormal behavior. This allows historical data to be intelligently summarized, preserving cross-stream correlation and flagging regions of interest that should be preserved in high detail for future reference. The techniques and benefits of `InteMon` are complementary to those provided by existing monitoring infrastructures, improving the types of (mis-)behaviors that can be flagged and improving the detail with which historical data is preserved.

Our prototype system allows these techniques to be evaluated in a real environment. It provides a web-based interface to allow a system administrator to view anomalies detected in the monitored data streams. It is currently monitoring a subset of the infrastructure in Carnegie Mellon's Data Center Observatory.

The rest of the paper is organized as follows: Section 2 gives a brief literature survey; Section 3 discusses the key ideas behind `InteMon`; Section 4 presents the architecture of our system; Section 5 illustrates the stream mining algorithm; Section 6 discusses some early experiences with the system as well as future work; Section 7 concludes.

2 Related work

There are a number of research and commercial monitoring systems, mainly focusing on system architecture issues such as scalability and reliability. Ganglia [18] 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 [19] is another hierarchical monitoring system which uses a custom kernel module running on each cluster node. ParMon [5] is a client/server monitoring system similar to ours but without mining capabilities. There exist commercial monitoring suites, such as OpenView [13], Tivoli [14], and Big Brother [4], as well as several open-source alternatives, including Nagios [16]. 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.

There is a lot of work on querying stream data, which includes Aurora [1], Stream [15], Telegraph [8] and Gigascope [9]. The common hypothesis is that (i) massive data streams come into the system at a very fast rate, and (ii) near real-time monitoring and analysis of incoming data streams is required. The new challenges have made researchers re-think many parts of traditional DBMS design in the streaming context, especially on query processing using correlated attributes [11], scheduling [3, 6], load shedding [10, 20] and memory requirements [2].

Here, we focus on the SPIRIT algorithm [17], 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 Main Idea

In this section, we present the main idea behind `InteMon`. In a nutshell, it tries to spot correlations and redundancies. For example, if the load on disk1, disk2, ... disk5 moves in unison (perhaps they are part of the same RAID volume), we want `InteMon` to spot this correlation, report it the first time it happens, and report it again when this correlation breaks (e.g., because disk2 starts malfunctioning).

The key insight is the concept of *hidden variables*: when all five units work in sync, they report five loads, but in reality all five numbers are repetitions of the same value, which we refer to as a "hidden variable." Correlations can also be more complicated (e.g., for a specific application, the disk load is a fraction of the CPU load). There can even be anti-correlations.

This viewpoint simplifies all three problems we mentioned in the introduction. Tracking a few, well chosen hidden variables allows for automatic configuration, anomaly detection, and compression:

Configuration: the human user does not need to know the normal behavior of a data stream: `InteMon` will learn it on the fly, and it will complain if there are deviations from it.

Reasoning: `InteMon` will report the timestamp and the numerical weights of the input data streams that caused the change in correlation. This provides the administrator with an ordered list of data streams that were involved in the anomaly, allowing him to focus on the most likely culprit.

Historical data: We can save a significant amount of space when storing historical data. First, there are fewer hidden variables than raw data streams, but there is still enough information to approximately reconstruct the history, because

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
memAvailReal.0	Available Memory
hrSystemNumUsers.0	Number of Users
hrSystemProcesses.0	Number of Processes
hrStorageUsed.1	Disk Usage

Table 1: Example SNMP metrics used by `InteMon`

there are redundancies and correlations. Second, since we know which timestamps were anomalous, we can store them with full information, compressing the rest of normal, "boring" data. This is analogous to compression for video surveillance cameras: during the vast majority of the time, things are normal, successive frames are near-identical, and thus they can be easily and safely compressed; we only need to store the snapshots of a few "normal" timestamps, as well as the snapshots of all the "anomalous" ones.

Next, we present the details of our implementation: the software architecture of our system and the user interface. In Section 5, we also present the mathematical technique to achieve on-line, continuous monitoring of the hidden variables.

4 System Architecture

In this section, we present our system design in detail. Section 4.1 introduces the real-time data collection process for monitoring sensor metrics in a production data center. Section 4.2 presents the database schemas for data storage. Then Section 4.3 shows the functionalities of the web interface.

4.1 Monitoring sensor metrics

Monitoring is done via the Simple Network Management Protocol (SNMP) [7]. SNMP was chosen because it is a widely used protocol for managing devices remotely, such as routers, hosts, room temperature sensors, etc. The large number of devices that support SNMP made it a natural place to start for data collection. However, any protocol that allows time-series data to be obtained could be used with `InteMon`.

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 Table 1) from all hosts in the data center using SNMP. At specific intervals, typically a minute, the server will query, via the snmpget program, each of the hosts and store the result in a customized MySQL database. Individual queries are spread out uniformly in the entire period to reduce the concurrent server load, and the load on clients is negligible. The streaming algorithms are then run across the incoming data to detect any abnormalities.

Table	Fields
MACHINE	id, type, name, address
SIGNAL_TYPE	id, properties, name, oid
STREAM	id, machine, signal_type
SPIRIT_INSTANCE	id, name, normalize_function
NORMALIZE_FUNCTION	id, name, function
INSTANCE_MEMBER	stream_id, spirit_id
RAW_DATA	stream_id, time, value
HIDDEN	hidden_id, spirit_id, time, value
RECONSTRUCT	stream_id, spirit_id, time, value
ALERT	spirit_id, time, alert_id, properties
ALERT_WEIGHT	alert_id, stream_id, weight

Table 2: Database tables used by `InteMon`

4.2 Database backend

In order to facilitate easily grabbing data via SNMP, the MACHINE table contains the host names of all the machines to be monitored, and the SIGNAL_TYPE table contains the OIDs of all the signals to be monitored. When the daemon runs, it performs a lookup in the STREAM table for all the streams that belong to each machine and queries the current value of each OID, via SNMP. The returned values are then stored in the RAW_DATA table, keyed by their stream_id and time. Because the STREAM table maps OIDs to machines, we have complete flexibility over which signals are monitored on each machine. SPIRIT_INSTANCE allows complete flexibility over which signals are grouped together for analysis. An entry in this table exists for each distinct set of streams that are analyzed together with a given normalization function that points to entries in a NORMALIZE_FUNCTION table. A NORMALIZE_FUNCTION entry is a function applied to the data before it is analyzed for correlations. There is also an INSTANCE_MEMBER table that maps each signal to the SPIRIT_INSTANCEs to which they belong. For example, to analyze correlations in network activity, a SPIRIT_INSTANCE could be created with all the network activity streams as members.

The data is then analyzed for correlations and the resulting hidden variables found are stored in the HIDDEN table, keyed by the SPIRIT_INSTANCE to which they belong, as well as the time. A change in the number of hidden variables indicates something anomalous is happening, causing the current correlations to break down. This triggers an alert which is stored into the ALERT table. The alert_id keys into the ALERT_WEIGHT table, which contains the relative weights of the signals that contribute to the new hidden variable. This provides an indication of what caused the correlation to break down, and it is useful for diagnosing the source of the problem. As a sanity check of the hidden variables, the original data is reconstructed from the hidden variables and stored in the RECONSTRUCT table.

4.3 Web interface

The JSP-based web interface is currently running on Apache Tomcat 5.5.15 with JRE 1.5.0. It consists of a main page with links to monitoring pages for each type of signal and each host. Also, this page lists the most recent alerts and the hosts/signals they affect as well as a link to a more extensive page of abnormalities. This provides the system administrator with the pertinent information that needs to be addressed immediately as well as tools to investigate further.

The individual monitoring pages consist of three graphs, shown in Figure 1. 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 signal monitoring pages, the first graph contains a minute-by-minute plot of all the signals of a given type, across hosts.

The second graph contains the hidden variables. For example, if all machines show the same pattern of CPU utilization, (e.g., a daily cycle), we have only one hidden variable, which is exactly a sinusoid-like wave with 24 hour period; now if half of the machines get overloaded to a 90% utilization, we need a second hidden variable, constant at 90%, to capture the fact. The system not only flags the abnormal timestamp, but also identifies the cause from association weights to the new hidden variable. In this case, CPU utilization has the largest association weight to the second hidden variable.

The last graph gives the reconstructed data. This graph uses only the hidden variables to try to approximate the original data, giving the user a feel for how well the algorithm is working. The host monitoring pages are similar, except they provide graphs of all signals monitored on a specific host. 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.

5 Stream mining

In this section, we describe the underlying mining algorithm in more detail. We follow standard matrix algebra notation for the symbols: Bold capital letters are matrices (e.g., $\mathbf{U}$); the transpose of a matrix is denoted with a T super-script (e.g., $\mathbf{U}^T$); bold lower case letters represent vectors (e.g., $\mathbf{x}$); normal lower case letters are scalars (e.g., n, k).

5.1 Correlation Detection

Given a collection of n streams, we want to do the following:

- Adapt the number of k main trends (hidden variables) to summarize the n streams.
- Adapt the projection matrix, $\mathbf{U}$, which determines the participation weights of each stream on a hidden variable.

More formally, the collection of streams is $\mathbf{X} \in \mathbb{R}^{T\times n}$ where 1) every row is a n-dimensional vector containing values at a certain timestamp and 2) T is increasing and unbounded over time; the algorithm finds $\mathbf{X} = \mathbf{Y}\mathbf{U}^T$ incrementally where the hidden variable $\mathbf{Y} \in \mathbb{R}^{T\times k}$ and the projection matrix $\mathbf{U} \in \mathbb{R}^{n\times k}$. In a sensor example, at every time tick there are n measurements from temperature sensors in the data center. These n measurements (one row in matrix $\mathbf{X}$) map to k hidden variables (one row in matrix $\mathbf{Y}$) through the projection matrix $\mathbf{U}$. An additional complication is that $\mathbf{U}$ is changing over time based on the recent values from $\mathbf{X}$.

Tracking a projection matrix: Many correlation detection methods are available in the literature, but most require O(n^2) comparisons where n is the number sensor metrics every time tick. This is clearly too expensive for this environment. We use the SPIRIT [17] algorithm to monitor the multiple time series. It only requires O(nk) where n is the number of sensor metrics and k is the number of hidden variables.

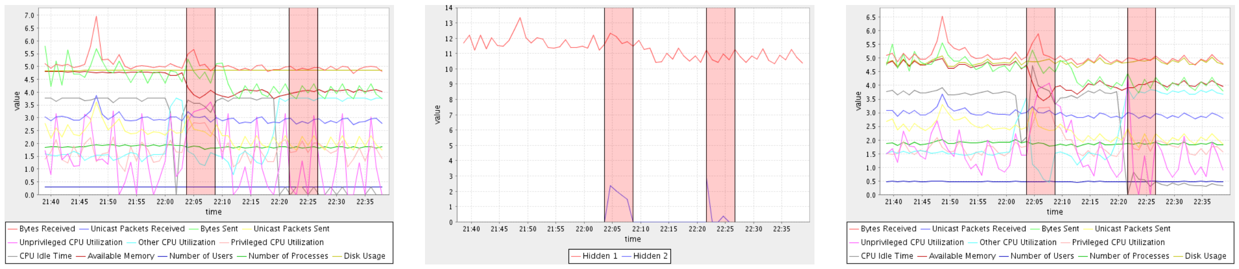
The idea behind the tracking algorithm is to continuously track the changes of projection matrices using the recursive least-square technique for sequentially estimating the principal components. To accomplish this, the tracking algorithm reads in a new vector $\mathbf{x}$ and performs three steps:

1. Compute the projection $\mathbf{y}$ by projecting $\mathbf{x}$ onto $\mathbf{U}$;
2. Estimate the reconstruction error ($\mathbf{e}$) and the energy (the sum of squares of all the past values), based on the $\mathbf{y}$ values; and
3. Update the estimates of $\mathbf{U}$.

Intuitively, the goal is to adaptively update $\mathbf{U}$ quickly based on the new values. The larger the error $\mathbf{e}$, the more $\mathbf{U}$ is updated. However, the magnitude of this update should also take into account the past data currently "captured" by $\mathbf{U}$. For this reason, the update is inversely proportional to the current *energy* (the sum of squares of all the previous values).

Detecting the number of hidden variables k: We use energy thresholding [12] to determine the number of hidden variables k. The idea is to increase or decrease the number of hidden variables when the ratio between the energy kept by the hidden variables and the one kept by the input values is below or above a certain threshold (e.g., 0.95 and 0.98 in our experiments).

For example, if we are monitoring the number of packets sent over the network across n hosts, the 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. Given the desirable energy threshold (e.g., 5%), the algorithm will



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

Figure 1: Web interface screenshots

pick the smallest number of hidden variables that satisfies the threshold.

Anomaly detection: We consider anomalies to be sudden changes of system behavior, that are indicated by a *change in the number of hidden variables*. More specifically, if the number of dimensions in this space changes, it signifies an abnormality. This sophisticated definition can capture anomalies beyond traditional threshold-based schemes, 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 present.

Missing values: Missing values can occur in the system, usually because SNMP uses UDP which is unreliable. In this case, we use the reconstruction values as a substitute. If no value is observed for an extended period of time, that host is marked as a dead node, which is also recorded as an abnormal event.

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

6 Discussion

This section discusses an early success with the `InteMon` system as well as shows how `InteMon` can help with automatic configuration and management of historical data.

6.1 Case study: Environmental monitoring

An early success of the `InteMon` monitoring system was to detect an anomaly in the environmental data from our data center.

A critical part of maintaining a high-availability data center is properly controlling the data center's environment (e.g., the temperature and humidity of the air). Many computer room air conditioning systems (CRACs) support remote monitoring of their operating status, and ours are no exception. The `InteMon` system monitors several parameters from the two CRACs in our data center. The monitored parameters are:

Return temperature: the temperature of the air as it returns to the CRAC to be cooled,

Supply temperature: the temperature of the air that is discharged from the CRAC into the room,

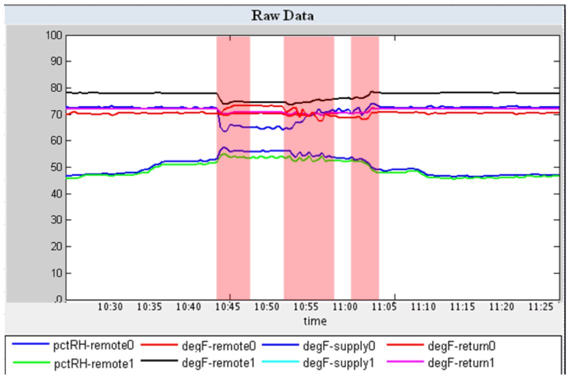


Figure 2: Temperature monitoring data

Remote temperature and humidity: the average temperature and humidity of the air across two remote sensors. These sensors are used by the CRACs to control the temperature and humidity in the data center.

The goal is to maintain the data center environmentals within the specifications for "class 1" environments as specified by ASHRAE [21, p10]. This specification provides a natural set of thresholds for conventional monitoring systems (e.g., temperature between 68 and 77 degrees Fahrenheit and 40% to 55% relative humidity). However, it is possible that the room needs attention even though both the temperature and humidity are within bounds.

Having a system that is able to detect anomalous fluctuations in the sensor data can alert the system administrator to trouble before the availability of the data center is impacted. One such incident was flagged by `InteMon`. The data in Figure 2 shows the measured sensor values. While the data was still within range, `InteMon` flagged the highlighted regions as anomalous. The altered pattern of data was caused by a problem with the supply air into the data center. The makeup air that was being introduced into the data center (to meet building occupancy requirements) was of such high humidity that the CRACs were over-cooling the air during dehumidification, triggering their internal reheat cycle. While this did not interfere with the functioning of the data center, it signalled a need to address the quality of the air entering the data center.

Although this condition could be detected by directly monitoring the makeup air or the status of the CRACs' reheat mode with a traditional threshold-based monitoring system, the need to do so (and the proper thresholds) were not obvious a priori. Instead, `InteMon` was able to detect the problem, reflected in other (monitored) sensor data and alert the administrator. Failure to detect and correct this problem leads to excessive power consumption within the data center and a potentially worsening of the problem until the environment can no longer be adequately controlled.

6.2 Historical data and space savings

When storing historical data, `InteMon` only stores the hidden variables and a small number of weight vectors (projection matrices) depending on the number of alerts. Compared to the raw data, it achieves a 10 to 1 savings when monitoring three machines with ten streams each. This savings is expected to be much larger when the number of streams is increased.

When it is necessary to access the historical data, it is reconstructed using the hidden variables and the projection matrix. The accuracy for that reconstruction is a configurable parameter, namely the *energy threshold*. In particular, to achieve the 10 to 1 savings, the threshold was 90%, meaning that the target is 90% accuracy during reconstruction.

7 Conclusion

We developed `InteMon`, an intelligent monitoring system targeting large data centers. `InteMon` operates on a real production cluster consisting of over 100 machines and about 250 TB of storage at Carnegie Mellon. The design goal for `InteMon` is to address three monitoring challenges: *configuration*, *reasoning*, and storage of *historical data*. `InteMon` uses efficient, state of the art algorithms to learn redundancies and correlations in the input streams of measurements.

For *configuration*, it learns what has been happening in the past, and it can flag deviations from the usual behavior. The human administrator does not need to figure out and specify what constitutes “normal” behavior.

For *reasoning*, `InteMon` reports not only what it considers anomalous, but also the “weight” matrix of PCA, which pinpoints the streams that caused the deviation.

For access to *historical data*, `InteMon` provides a natural way to compress historical data adaptively. When many input streams are correlated, `InteMon` stores only one copy of them, and it records the scaling factors that are needed to reconstruct the rest. Moreover, it pinpoints the timestamps of the anomalies. Thus, when compressing historical data, `InteMon` can adaptively compress the “normal” intervals, while spending more emphasis and disk space on anomalies.

In addition to its technical strengths, `InteMon` has been carefully designed with an intuitive graphical web front-end and an interface to a relational database management system (MySQL) for data storage.

This is a joint effort to bridge the data mining and the operating systems community; we believe that such collaboration is crucial to our efforts to build self-monitoring, and eventually self-organizing, data centers.

References

[1] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[2] A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing memory requirements for queries over continuous data streams. In *PODS*, 2002.

[3] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain : Operator scheduling for memory minimization in data stream systems. In *SIGMOD*, pages 253–264, 2003.

[4] Big Brother. http://www.bb4.org.

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

[6] D. Carney, U. Cetintemel, A. Rasin, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Operator scheduling in a data stream manager. In *VLDB*, 2003.

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol (SNMP). RFC 1157, Network Working Group, 1990.

[8] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[9] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *SIGMOD*, 2003.

[10] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD*, pages 40–51, 2003.

[11] A. Deshpande, C. Guestrin, S. Madden, and W. Hong. Exploiting correlated attributes in acqusitional query processing. In *ICDE*, 2005.

[12] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 2nd edition, 1990.

[13] HP OpenView. http://www.managementsoftware.hp.com/index.html.

[14] IBM Tivoli. http://www.ibm.com/software/tivoli/.

[15] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR*, 2003.

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

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

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

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

[20] N. Tatbul, U. Cetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB*, 2003.

[21] TC9.9 Mission Critical Facilities. *Thermal Guidelines for Data Processing Environments*. ASHRAE, 2004.