Xmon: A Lightweight Multilayer Open Monitoring Tool for Large-scale Virtual Clusters

Cong Yang, Jue Hong, Cheng-Zhong Xu, Wen-Long Du, and Dong Lin
Cloud Computing Research Center, Shenzhen Institutes of Advanced Technology
Chinese Academy of Sciences
Shenzhen, P.R.C China
{cong.yang, jue.hong, cz.xu, wl.du, dong.lin}@siat.ac.cn

***Abstract*—Metrics coming from virtual clusters can be used for task and node errors locating, diagnosis and predicting. They are crucial for management and performance measurement of large-scale clusters. However, conventional monitoring system has limited ability of multilayer cluster metrics collection, cluster diagnosis and automatic error correction. In this paper, we present Xmon to settle previous problems. Specifically, Xmon uses a range of /proc and commands parsing methods and the open API Node Daemon to collect multilayer metrics includes process, virtual and physical layers. Secondly, by Decision Tree, Diagnosis Module of Xmon can generate a cluster report which includes an errors list and a cluster score to help managers quickly realize and locate running errors in virtual and physical clusters. Finally, the Controlling Unit with opening API in the Node Daemon Agent can change the running state of problematic process automatically and push alarming messages to managers. With the support of NoSQL, AMQP, RPC and other technologies, the performance, scalability and reliability of Xmon are effectively ameliorated. Evaluation shows that Xmon is able to collect metrics from physical and virtual node effectively with few segment faults. We also show that Xmon loads more than 1000 nodes with just 0.065% of the total network's bandwidth and no more than 10 packages lost in one hour.**

Keywords-Cluster Monitoring; Cloud Computing; Virtural Machine; NoSQL; Decision Tree

I. INTRODUCTION

As a high-level virtual system component, virtual machine provides a more advanced abstraction for resources [10]. Virtual cluster is composed by virtual machines, on which running the same tasks. Virtual Cluster designed on the basis of virtual machine and network to provide virtual executing environment for large-scale distributed applications. Virtual clusters offer the fundamental support for brand-new cloud computing systems as well as virtual High Performance Computing system.

The cluster monitoring system observes the behavior, running status and other vital information of nodes and clusters, stores and analyses metrics, and displays results. It is crucial for cluster management and performance measurements as the monitoring data can be used to diagnose problems and to suggest remedies by both end users and system administrators.

However, traditional monitoring tools have limited abilities of process and virtual layer metrics collection [2, 3, 6, and 7], automated errors analysis and mining process [3], virtual cluster and task monitoring, open API, etc. Some of monitoring systems produce too much pressure to the both node and cluster [2]. All of these drawbacks motivated us to design and develop a lightweight multilayer tool for large-scale virtual and physical clusters.

This paper proposes a new monitoring system, named Xmon, which solves aforementioned problems by a novel architecture of system and Node Daemon, data processing and algorithm of metrics mining. The architecture of Xmon is composed by three components, in which there are several sub-components which could be flexible gathered or divided onto different services based on the scale of clusters. We use RPC technology and open API methods to ensure the lightweight and adaptively of Xmon Node Daemon. We also provide the structure of Decision Tree and the process of cluster health diagnosis.

We make the following contributions:

- We experimentally reveal an automatic cluster health diagnosis module over the incoming streams and storage metrics.
- We propose a new architecture of monitoring tool for large-scale virtual and physical clusters. It achieves effects of real-time monitoring, MapReduce metrics analysis, alarming, etc.
- We propose architecture of lightweight Node Daemon which collects metrics in different layers and supports redevelopment based on its open API.

The rest of the demo proposal gives a brief literature review in Section 2. We present the architecture of Xmon in Section 3. We discuss the detailed implementation of Xmon architecture, cluster diagnosis algorithm and visualization in Section 4. System evaluation, overhead and scalability, cluster diagnosis module evaluation are discussed in Sections 5. We conclude in Section 6.

II. RELATED WORK

Before starting to design the Xmon architecture, we analyzed many documents about state-of-the-art monitoring systems. However, most of these systems have some limitations in monitoring layers, process, architecture, etc.

Therefore, in this paper, we concentrate on the following issues:

1) Monitoring layers: with the development of grid virtualization and cloud computing, more and more virtualization products are used in the cloud computing platform. However, traditional tools just collect the vital metrics of physical machine and have little ability of task and virtual machine information collection.

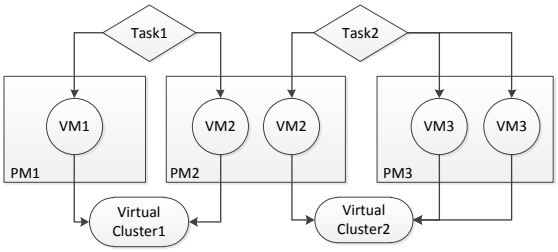

Figure 1. Virtual Cluster (PM=physical machine, VM=virtual machine)

As is shown in Figure 1, when users run tasks with virtual machine in the cloud computing cluster, every virtual machine which runs the same task will be organized as a virtual cluster. Users need to know the task process and the status reports of virtual cluster. Therefore, Xmon is designed to collect multilayer metrics which not only includes process and physical machines, but also virtual machines and virtual clusters.

2) Monitoring Process: conventional monitoring tools such as Ganglia [2] and Supermon [3] collect metrics via pushing or pulling mechanism from each node at an interval that allows new data to be available and the monitoring system to keep up effectively. However, they do not provide automatic analysis and any data mining process. In other words, they are not smart. They mainly provide administrators with the ability to view the primitive data on a per-node basis. Typical commercial management tools, such as those from IBM Tivoli [4] and HP OpenView [5], compare these instantaneous data values on a per-machine basis to predefined thresholds and either send notification to the system administrator or automatically shut down or reboot the system in response. Accordingly, other than the main features of monitoring, Xmon also needs the control unit in the node daemon agent.

3) Monitoring architecture: most existing monitoring systems in clusters maintain a centralized server or a set of hierarchically organized servers to aggregate and index monitoring system performance data, such as Supermon [3], PARMON [6], ClusterProbe [7] and R-GMA [8], which use a centralized server to monitor system performance. In contrast, Globus MDS2 [9] and Ganglia [2] employ a set of hierarchical Servers. Nonetheless, the problem is that the centralized server might become both a bottleneck and a single point of failure in large-scale cluster environment. For the hierarchical system, the partitioning scheme is often predefined and cannot adapt to the dynamic changes of the monitoring nodes. For example, if the node running Ganglia *gmetad* is under heavy load and can't reply monitoring data in time, the upper level *gmetad* has no idea of the situation and will wait for a long time. Therefore, Xmon architecture is disrupted, and the main Service is composed of 4 independent components which can be deployed in one single service or the service cluster according to the nodes quantity.

4) Metrics analysis: several monitoring systems have been developed mainly to settle the issues of scalability and reliability [10, 11, and 3]. And most of them provide a graphical interface for viewing the cluster behaviors in real time and also log the raw monitoring data. However, there is no such manpower to make effective use of this real-time monitoring and most of the useful but hidden information has not been discovered. [11] Thus, an efficient data mining module is absolutely crucial for monitoring systems, and this capacity is essential for turning the passive monitoring tool into an active administrator assistant. Xmon is an intelligent monitoring system that is being built to address these issues. The data mining and cluster diagnosis module of Xmon should be able to analyze the monitoring data in real-time and report the patterns and anomalies for inspection. In particular, the control unit of Xmon node daemon can turn the process state timely when the diagnosis module finds errors in the running process.

III. ARCHITECTURE OF THE XMON

In this section, we present Xmon design in details. Firstly, we discuss the Xmon Node Daemon architecture and Metrics Delivery process. After that, we introduce the architecture and working process of Center Service. Meanwhile, we also talks about the NoSQL Database Service, which is responsible for the data storage and cluster health diagnosis. Final section displays shows the GUI of Xmon in the web and Android Client. The overall system design is shown in Figure 2.

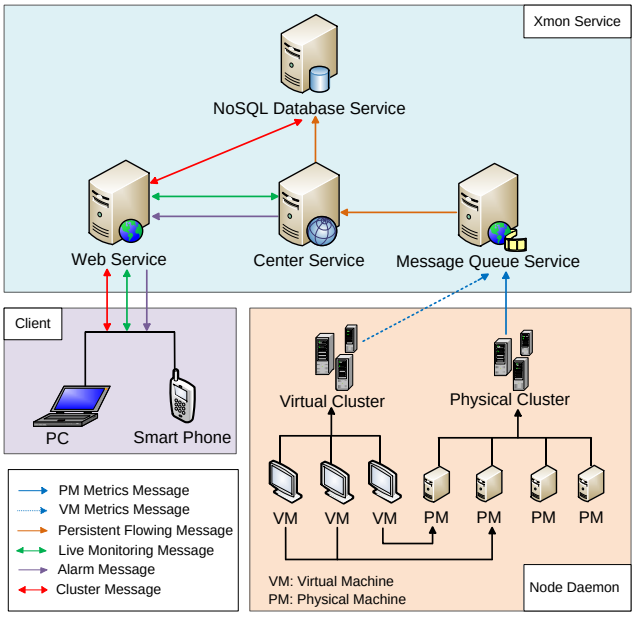

Figure 2. Architecture of Xmon.

A. *Xmon Node Daemon and Metrics Delivery*

Xmon Node Daemon is mainly in charge of metrics collection, packaging, and sending. As is shown in Figure 3, node daemon collects metrics from different layers, which could meet different demands from different monitoring requests. Xmon Node Daemon is an opening environment which enables people to redefine it according to their individual requirement.

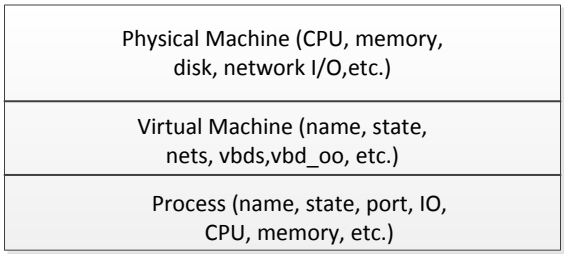

Figure 3. Metrics that be Collected in Different Layers.

Node Daemon is applied to scan and parse node information periodically, and send the latest packaged metrics to the Message Queue Service with the frequency predefined by the users. When Node Daemon is killed or the service is rebooted, it will restart automatically.

After metrics are collected and packaged, it will be sent to the Message Queue Service. Message Queue Service is responsible for monitoring metrics delivery and exchange, using AMQP as the basic queuing and routing protocol. The Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for message-oriented middleware. The defining features of AMQP are message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security. [12]

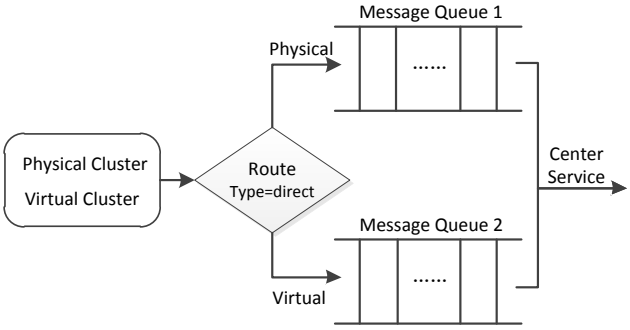

Figure 4. Message Queue Service.

As is shown in Figure 4, Xmon Message Queue Service divides the original messages into two types in terms of the Route. Each type is delivered into the Message Queue, waiting for the Center Service requests. To achieve the AMQP functions, we use rabbitMQ as the Message Queue Service's main broker software. RabbitMQ is an open source message-oriented middleware, using the standard Advanced Message Queuing Protocol (AMQP). The RabbitMQ server is written in Erlang and is built on the Open Telecom Platform framework for clustering and failover. [13]

B. *Center Service and Data Storage*

Centre Service is responsible for data streaming analysis, Database Service and Web Service communication, commands transfer and Node List updates.

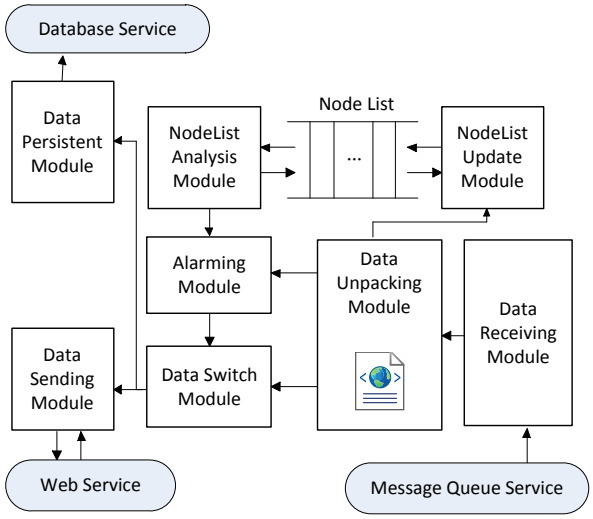

Figure 5. Architecture of Center Service.

Figure 5 is the architecture of Xmon Center Service, which is composed of the following main components:

1) Data Storage Module: this module is in charge of communicating with NoSQL Database, metrics and system log insertion. Data Storage Module is a thread of Center Service process, which is always running after the Center Service is started.

2) NodeList: NodeList is a Hash Table which stores the latest basic node information. Table 1 is the NodeList form, in which we can see that nodes are distinguished by several marks including federation name, cluster name and cabinet name, etc. In the Xmon Center Service, NodeList is visited and updated with very high frequency; hence, it needs to be placed in the memory of Center Service machine with hash table.

TABLE I. NODELIST FORM

Name	Describe	is NULL
ID	ID number of Node	no
Federation	federation name	yes
Cluster	cluster name	no
Cabinet	cabinet name	yes
Node Name	node name	no
Node IP	node ip (eth0 prior)	no
Last Update	last update time (S)	no

3) NodeList Update and Analysis Module: this module is in charge of the NodeList updating and analyzing, which is also a non-stop thread of the Center Service process. When the new package is collected and unpackaged, node mark will be extracted and analyzed by NodeList Update Module

to check whether this node exists in the NodeList. If it exists, the node information in the NodeList needs to be updated. If not, the new node information will be inserted into the NodeList and meanwhile, the Alarming Module will send the alarming message to the Web Service.

What's more, Analysis Module periodically analyzes the NodeList to check the overdue node information. If the duration between the current date and the date of the latest update in the NodeList surpasses the time defined by the users (default time is 10s), the Analysis Module will drops this node and send the alarming message to the Alarming Module at the same time. These two modules can keep the NodeList fresh and up-to-date.

4) Alarming Module: this module is another non-stop thread of the Center Service process which is responsible for data checking, listening and organization. Alarming module is to examine the unpackaged data stream uninterrupted. Once metrics triggers the alarm condition, Alarming Module will organize and push alarming messages immediately.

All in all, Center Service is a robust process which includes three non-stop threads that are responsible for different duties of data processing and analyzing. Center Service loads parameters from XML file including node data sending frequency, Web Service IP address, NoSQL Database Service IP address and Port, etc.

NoSQL Database Service is responsible for metrics storage, cluster diagnosis, and log storage. It is widely acknowledged that monitoring metrics come from different layers and environment, like Windows and UNIX. Therefore, most of them are unstructured and loosely-related and it's hard to store and analyze them by SQL Database. Therefore, using NoSQL database system can easily solve this problem. NoSQL means "not only SQL" [14], which is developed to manage large volumes of data that do not necessarily follow a fixed schema. NoSQL database management systems are useful when working with a huge quantity of data and the data's nature does not require a relational model for the data structure. The data could be structured, but it is of minimal importance and what really matters is the ability to store and retrieve great quantities of data, but not the relationships between the elements.

In Xmon, we use mongoDB as the NoSQL database service system. MongoDB (from "humongous") is an open source document-oriented NoSQL database system [15]. Written in C++, mongoDB is part of the "new" NoSQL family of database systems. Instead of storing data in tables as is done in a "classical" relational database, mongoDB stores structured data as JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster.

C. GUI

Web Service supplies data pool and real-time service to the JSP and Android based client to support GUI for Xmon. Written in Java, web service on the one hand restlessly updates Data Pool by communicating with Center Service to provide the fresh monitoring metrics to the client and communicate with Database Service to supply cluster diagnosis results on the other hand.

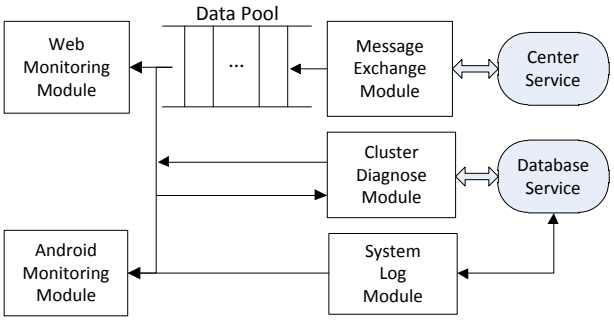

Figure 6. Web Service.

As is shown in Figure 6, Data Pool actually is a static *ConcurrentHashMap* which stores the latest metrics of the whole cluster. System Log Module is a XML file located in the same root directory of Xmon. With RPC technology, Message Exchange Module is responsible for the message receiving and the Data Pool updating. Cluster Diagnose Module is used in the agent between the Database Service and the client.

Clients include Web page client and Android client. The Web page client mainly covers cluster (physical and virtual) information, status and diagnosis, virtual and physical node real-time monitoring, alarming, etc. Android client is mainly responsible for alarming, cluster and node static information searching. Visualization and interaction of client will be discussed in next Section.

IV. IMPLEMENTATION

The goal of Xmon is to exploit a lightweight multilayer tool for large-scale virtual clusters. Therefore, we will discuss how to realize lightweight, multilayer monitoring and open API in Section A. After that, we provide the process and architecture of cluster diagnosis module in Section B. Finally, real-time visualization will be discussed in Section C.

A. Lightweight and Open API Xmon Node Daemon

Before starting to design the architecture of Xmon Node Daemon, we had confirmed that this component at least achieves the following goals:

- Multilayer metrics collection.
- Opening API, supporting redevelopment.
- As lightweight as possible.

To achieve the goals mentioned above, we have designed the architecture of Xmon Node Daemon, as illustrated in Figure 12. Written in C++, Node Daemon has 3 important units:

1) Data Collection Unit: There are 3 modules in this unit. */proc analysis module* is the default module, which is used for /proc scanning and parsing. In this module, useful metrics are collected and send to the Data Packaging Unit. For virtual machine cluster, we collect both process information

(PID, state, port, vmsize, cpus_allowed, stack usage, etc.) by /proc parsing and virtual machine information (CPU and memory usage, Nets, etc.) by running different commands like *xentop -bi1*, etc. By opening API, User Defined Collector Module is responsible for users to define their own collecting methods, such as single layer and metric collecting, etc. Third-part Collector Module is used in third-party plug-ins collection tools (or protocols) using or configuring, such as SNMP, CPU fan speed and temperature plug-ins. With this unit, users can conveniently call or define data collection methods.

2) Data Packaging Format: The default packaging method is JSON (JavaScript Object Notation), a lightweight text-based open standard designed for human-readable data interchange. Other Packaging Module and User Defined Packaging Module are opening-API modules. Working with two opening-API modules in the Data Collection Unit, these 4 modules ensure the flexibility and applicability of the Xmon Node Daemon.

After metrics were packaged, it will be queued by the Message Queue Service. After that, all of these metrics are converging in the Center Service, waiting for examining, storing, and exchanging. Monitoring metrics processing is shown in Figure 7.

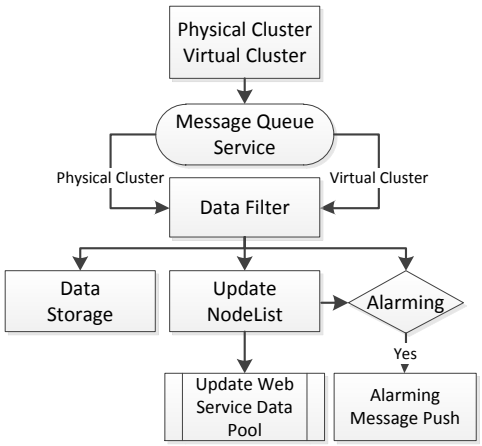

Figure 7. Data Processing in Xmon.

Before packaged metrics flows into the Message Queue Service, each metric should be marked with different marks based on its layer. Mark is composed of the following elements:

$$mark = metrictype + nodename + nodeip + cabinet + cluster + federation$$

For example, if a metric comes from physical machine, the mark should be:

PHC-ccrfox8-202.198.129.254-CA34-XJYCM-SCGN

If it comes from a virtual machine, it should be:

VIC-domain0-202.198.129.233-CA35-XJYVCM-SCGN

As is shown in Figure 7, marked metrics flows into the Message Queue Service, and is divided and pushed into two different queues by its marks. After that, queues are filtered and collected in the Center Service. In Center Service, clean metrics are sent to three non-stop threads at the same time for storage, NodeList updating and alarming

3) Controlling Interface:

We believe that a good monitoring system is not only used for cluster monitoring, but also for controlling. However, too much control functions in the node daemon might take up more node resources. Therefore, based on RPC (Remote Procedure Call) technology, we design the Control Unit to make Node Daemon more lightweight and intelligent.

- Sleeping Function. This section is to ensure that the daemon lightweight running. Some non-commonly-used thread is not in the running state, but in a "sleeping" state, waiting for the Data Receive Module to receive commands through RPC and activate it. These threads include daemon pausing, sending frequency updating, process tracking, etc. By this module, even node daemon process is full of functions, but with small usage of resources.
- Process Control. This section is responsible for changing the running state of a process according to the instructions of the Xmon Center Service. The significance of this module is to help nodes running more securely by locking the running status of doubtful process before the alarming message reaches the clients.
- User Defined Control. Similar to units, users can define their own control functions by the Opening API in this module.

Except for these 3 modules, Xmon Node Daemon also has Listening Module and System Parameter XML, Listening Module is responsible for commands listening and data pushing by RPC technologies and rabbitMQ data pushing functions, respectively. System Parameter XML is used as system configuration file. Each Services and Node Daemons share the same System Parameter XML that ensures the same frequency, contents and forms of metrics. Structure of the System Parameter XML is illustrated in Table 2.

TABLE II. CONTENTS OF SYSTEM PARAMETER XML

Parameter	Describe
Web Service IP and Port	String and int, respectively
Center Service IP and Port	String and int, respectively
Message Queue Service and Port	String and int, respectively
DataBase Service IP and Port	String and int, respectively
DataBase Name and Collection	String and String, respectively
Default Send Metrics Frequency	int, default is 5s
Monitoring Layers	Physical, Virtual, Process. bool
SystemType	Windows, Linux, etc. bool
Default Packaging Type	JSON, bool
CollectionType	/proc, user defined, third-part, bool
Control Type	process control type, bool
Metrics List	40 metrics list, bool in each

B. *Cluster Diagnosis*

Cluster Diagnosis module is a non-stop thread running in the Database Service which makes Xmon more intelligent and useful.

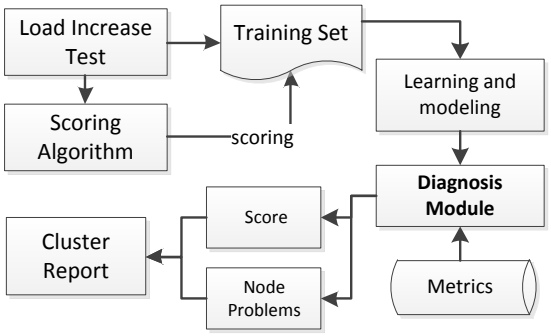

Figure 8. Principle of cluster diagnosis module.

The mainly mission of the Cluster Diagnosis is to help users to manage nodes or clusters more effectively and conveniently by reading the diagnosis reports. Figure 8 shows the principle of the cluster diagnosis module. Cluster report is composed of Score and the node problem list.

Score is set between 0 and 100, describing the general health condition of the cluster. For example, Score = 0 means that all nodes in cluster are dead or shut down. Score = 100 means that there is no task or virtual machine running on the cluster. Higher than 60 scores means that virtual and physical cluster is running stable with no more than 4 errors or warnings in the node problem list. Lower than 30 scores means that some nodes in the cluster are unstable or dead, and there are more than 6 errors or warnings in the node problem list. By checking the cluster score and the node problem list, users can quickly be informed of the general cluster status and problems. We use Decision Tree as the basic diagnosis module. Attribute Selection, Final Score and The relationship between Attributes and the final Score are the primary aspects for research.

We deploy the cluster diagnosis unit on the NoSQL Database Service. To support large-scale metrics MapReduce analysis, we are fully making use of the mongoDB master-slave replication features, setting up a master node and 3 slaves (Figure 9).

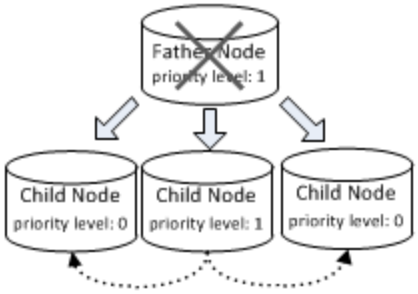

Figure 9. NoSQL Database cluster.

MongoDB supports an automated sharding/partitioning architecture, enabling horizontal scaling across multiple nodes. For Xmon that outgrow the resources of a single database server, mongoDB can convert to a sharded cluster, automatically managing failover and balancing of nodes, with few or no modifications to the original application code. As is shown in Figure 8, a replica set is a set of three servers, each of which contains a replica of the entire data set for the given shard. One of the n servers in a replica set will always be primary. If the primary fails, the remaining replicas are capable of electing a new master according to the predefined priority level (dotted line). Details of diagnosis module are discussed in technical report [17].

C. *Real-time Visualization*

To realize the real-time monitoring, we defined a static Hash Table, in which stored the latest metrics of clusters by JSON format. Hash Table is continuously updated by the Center Service with RPC and visited by web page by Highcharts and Servlet. Highcharts is a charting library written in pure JavaScript, offering intuitive, interactive charts to web site or web application. [16]

The user interface view is shown in Figure 13. Xmon uses Highcharts to visualize historical and real-time monitoring information for cluster (physical and virtual), node (physical and virtual), and metric trends over different time granularities ranging from minutes to hours. Highcharts is a popular middleware for graphing time series data. For cluster general status, Highcharts generates graphs which plot historical trends of metrics versus time. The default time is one hour, but users can define the time granularity according to their needs. These graphs are then used by Xmon and exported to users using a JSP web front-end.

Because of the opening API and JSON format metrics, Xmon can use third-part tools to visualize historical monitoring information like RRDTool (Round Robin Database), etc. This allows website developers to easily customize the look and feel of the website without damaging the underlying content engine.

Users can conveniently interact with Xmon web page by choosing and adjusting clusters, nodes, metric types, frequency, chart size, etc. What's more, managers also can check system logs and cluster status by cluster diagnosis module. They can send commands to the Center Service and Node Daemon by the systemparameter page.

V. EVALUATION

In this section, we present a quantitative analysis of Xmon along with an account of experience gained through real world deployments on SIAT Clusters. For the analysis, we measure scalability and performance overhead in both Xmon Node Daemon and global. Also, we measure the package loss rate in different components of Xmon to find the bottleneck of Xmon.

A. *Experiment Setup*

In this paper, we used Xnebulae cloud computing systems and normal PC cluster to evaluate Xmon. Xnebulae is a cluster in the Cloud Computing Research Center, Shenzhen Institutes of Advanced Technology (SIAT) which consists of approximately 15 SMP nodes, each consists of 16

2.40GHz Genuine Intel Xeon(R) E5620 CPUs, 6 GB of RAM, Four 1 TB disks, Gigabit Ethernet connections and run the SUSE Linux Enterprise Server 11 (x86_64) system.

The normal PC Cluster is Lenovo table PC Ubuntu cluster in the Cloud Computing Research Center in SIAT, currently be used for web-based system testing. The cluster consists of approximately 5 dual-processor SMP nodes. Each SMP is a Lenovo M5500 desktop, each of which contains AMD 2.70GHz Athlon (tm) 7750 CPU, 2.00GB memory and one 500GB disk. The cluster also includes a uses Extreme Networks Huawei switches for Gigabit I/O connectivity between the nodes. All nodes in the cluster run the Ubuntu 10.04 LTS-64 bit.

B. *Overhead and scalability*

Before a cloud computing monitoring system to be widely used, it must first meet the prerequisites of having low performance overhead and being able to scale to cluster size systems. To quantify this, we performed a series of experiments on clusters running Xmon. Because each default Node Daemon is responsible for metrics collection and sending, and all of the metrics are collected by Message Queue Service. Therefore, for performance overhead, we measured both local overhead incurred within the nodes (e.g. CPU overhead, memory, network bandwidth) as well as ‘‘global’’ overhead incurred between the nodes and Xmon Service. For scalability, we measured overhead on individual nodes and quantified how overhead scales with the size of the system, both in terms of number of nodes within a cluster and the number of services being Xmon Service.

1) Xmon Node Daemon Overhead: In Table 3, we show local per-node overheads for local monitoring for Xnebulae and personal PC cluster. Data for this table was collected by running the *top* command every 5 seconds to obtain process information and averaging the results. For Xnebulae, these numbers represent the per-node overheads for a cluster of 15 SMP nodes. For normal PC Cluster, these numbers represent the per-node overheads for a cluster of 5 SMP nodes.

TABLE III. NODE DAEMON OVERHEAD

Clusters	Obtained Items (Average)		
	CPU Usage (%)	*Phy Mem (kb)*	*network out(byte)*
Xnebulae	< 0.1	84	860
PC Cluster	< 0.1	96	660

Local per-node overheads on Xnebulae and normal PC cluster are quite small. Per-node overheads for nodes at both Xnebulae and PC cluster account for less than 0.1% of the CPU, respectively. Physical memory usage is moderate, 84kb and 96kb for Xnebulae and PC cluster respectively. Network out, on the other hand, are small. On Xnebulae, Xmon Node Daemon has only 860bytes metrics produced in each period of data collection in single node. On PC cluster, the number is even smaller, just 660bytes per-period of data collection on each node. Since Xmon Node Daemon only collects physical machine information, no I/O overhead is incurred.

2) Global overhead: On each node of Xnebulae and PC cluster, we deployed virtual machines at random, but no more than 16 and 2 on Xnebulae and PC cluster, respectively. Virtual machine is chosen from Virtual Machine List in table 4 at random.

TABLE IV. VIRTUAL MACHINE LIST

ID	*VCPUS*	*Memory (max, KB)*	*NETS*	*VBDS*	*NETTX (max, KB)*	*NETRX (max, KB)*
1	1	2097000	1	3	97500	27900
2	2	1097000	1	1	97500	27900
3	1	2097000	1	2	975000	279000

To measure the global overhead of Xmon, we decompose the network bandwidth into physical cluster monitoring bandwidth and virtual cluster monitoring bandwidth. All bandwidth information is collected by Xmon Queue Service, in which metrics are divided into two different queues (Physical and Virtual) and are measured by default polling interval. By writing queued messages into two different files, we can easily record file size every 5 seconds and then summarize global overhead.

TABLE V. GLOBAL OVERHEAD

Clusters	Obtained Clusters	
	Physical Cluster(KB/5s)	*Virtual Cluster(KB/5s)*
Xnebulae	9.75	3.69
PC Cluster	2.15	1.07

As is shown in table 5, global overheads on Xnebulae and normal PC cluster are also quite small. Virtual cluster overhead is based on the number of task and virtual machine, but only takes limited bandwidth.

3) Scalability: In these experiments, we characterize the scalability of Xmon as we scale the number of nodes within a cluster. To measure scalability within a single cluster, we use the PC cluster. We selectively disable Xmon Node Daemons to obtain cluster sizes ranging from 1 to 1000 Node Daemons and measure performance overheads. By this method, each node runs no more than 200 Daemons in PC cluster.

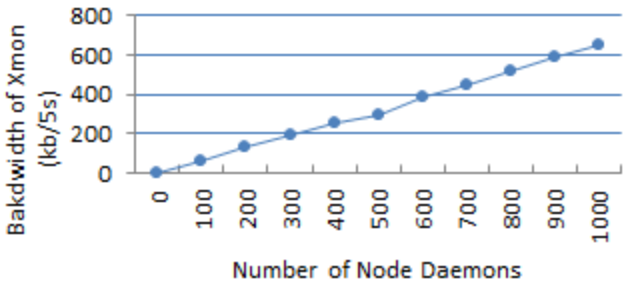

Figure 10. Network Bandwith of Xmon.

In Figure10, we quantify the scalability of Xmon on a single cluster by showing the total Message Queue Service bandwidth consumed as a function of cluster size. By recording bandwidth in Message Queue Service, we observe a linear scaling in bandwidth consumed as a function of cluster size. In this case, we observe small constant factors, which can be at least partially attributed to rabbitMQ's use of thresholds.

At 1000 node daemons, for example, we measure bandwidth consumed to be just 37.32Kbits/5s. On a Giga bit Ethernet network, 652.32 Kbits/5s amounts to just 0.065% of the total network's bandwidth.

Reliability: In this section, we use PC cluster to evaluate the reliability of Xmon. The reliability testing of the overall system is based on the package loss rate in different components of Xmon when the number of Xmon Node Daemon keeps increasing. We increase the number of Node Daemons from 10 to 1000, which sampling points 10, 100, 200, 300, 400, 500, 600, 700, 800, 900 and 1000. During this time, we count totall packages number in nodes, Message Queue Service, Database Service and web page every 1 hour. After samples are collected, we figure out the number of dropped packages in different components. Through comparative analysis of these four components, the comprehensive evaluation of the system bottleneck and overall reliability will be worked out.

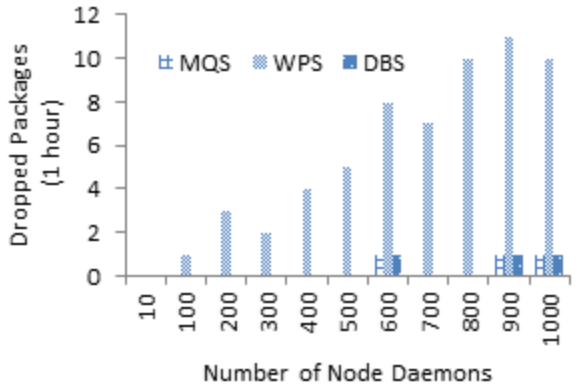

Figure 11. Dropped package number in each components.

As is shown in Figure 11, MQS and DBS mean a drop in packages number in Message Queue Service and Database Service, respectively. After comparing the number of dropped packages, we find that it is almost the same in these two components, which suggests that the packages loss in DBS is mainly due to packages dropping in MQS. But the decline is negligible and in the acceptable range.

WPS means dropped packages number in Web Page which is the bottleneck of Xmon. Figure 11 shows that when the number of Node Daemon rises to 600, more and more packages are dropped. We find that the drop of packages in web page is due to Highcharts and its way of getting data from Web Service. One Highcharts in Xmon produces one servlet, and too much Highcharts will create the same number of servlets in one web page which would reduce the efficiency charts updates and lead to the package loss. Therefore, this component needs to make further improvements to reduce the package loss rate.

VI. CONCLUSION

In this paper, we presented the architecture, implementation, and evaluation of Xmon, A lightweight multilayer open monitoring tool for large-scale physical and virtual clusters. Xmon is based on an inattentive hierarchical design which uses RPC listen/announce protocol and the emerging AMQP standard to monitor state within clusters. It uses a careful balance of simple design principles and sound engineering to achieve high levels of robustness and ease in management. Based opening API on Node Daemon, the implementation has been ported to an extensive set of operating systems and processor architectures and is currently in use on Xnebulae cluster in SIAT. With parsing, collecting and gathering virtual machine and task information, Xmon realizes the function of virtual cluster and virtual machine monitoring. According to experiments, the monitoring system only takes a small amount of resources of nodes and clusters, with high efficiency. The actual practice demonstrates that the monitoring system has good stability. Xmon has been running steadily on Xnebulae with little decline in performance.

Future work will mainly focus on the Node Daemon architecture and stability to improve the scope and security of Xmon.

ACKNOWLEDGMENT

This material is based upon work supported by the Shenzhen Institutes of Advanced Technology (SIAT). Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Chinese Academy of Sciences, or other funding parties.

We do appreciate the invaluable data and discussion offered by Man-Li Zhou from Northeast Normal University and Yan-Yi Wan from Hong Kong University of Science and Technology.

REFERENCES

[1] J. M. Brandt, A. C. Gentile, D. J. Hale, and P. P. P´ebay, "OVIS: A Tool for Intelligent, Real-time Monitoring of Computational Clusters,". Parallel and Distributed Processing Symposium (IPDPS 06), pp.25-29.

[2] Matthew L.Massie,Brent N.Chun and David E.Culler, "The ganglia distributed monitoring system: design, implementation, and experience,", Parallel Computing 30 (2004) , pp.817–840.

[3] Matthew J.Sottile and Ronald G.Minnich, "Supermon: A high-speed cluster monitoring system,", Cluster Computing, 2002. Proceedings, pp. 39–46.

[4] "IBM Tivoli", http://en.wikipedia.org/wiki/Tivoli_Gardens.

[5] "HP OpenView", http://en.wikipedia.org/wiki/HP_OpenView.

[6] R.Buyya,"PARMON: a portable and scalable monitoring system for clusters,", Software: Practice and Experience,vol.30, pp.723-739,2000.

[7] Z.Liang, Y Sun, and C Wang. "ClusterProbe An Open, Flexible and Scalable Cluster Monitoring Tool", In IEEE Intercational Workshop on Cluster Computing, pp. 261-268,1999.

[8] A.W.Cooke, "The Relational Grid Monitoring Architecture:Mediating Information about the Grid", Journal of Grid Computing, vol. 2, no. 4, December 2004. pp.1-28.

[9] S. Czajkowski, K. Fitzgerald, I. Foster and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", Proc.of HPDC,2001. pp. 1-14.

[10] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, "A case for grid computing on virtual machines", in Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003.

[11] Evan Hoke, Jimeng Sun and Christos Faloutsos, "InteMon: Intelligent System Monitoring on Large Clusters," Proc. IEEE Symp. Proceedings of the 32nd international conference on Very large data bases (VLDB'06), IEEE Press, September 12-15, 2006, pp. 1239-1242.

[12] O'Hara, J, "Toward a commodity enterprise middleware", Acm Queue May/June 2007, Vol.5 No.4. pp. 48–55.

[13] "rabbitMQ", http://www.rabbitmq.com/

[14] "NoSQL", http://nosql-database.org/.

[15] "mongoDB", http://en.wikipedia.org/wiki/MongoDB.

[16] "Highcharts", http://www.highcharts.com/.

[17] Cong Yang, Cheng Zhong-Xu, etc. "Research on the Health Diagnosis Module of Large-scale Clusters".

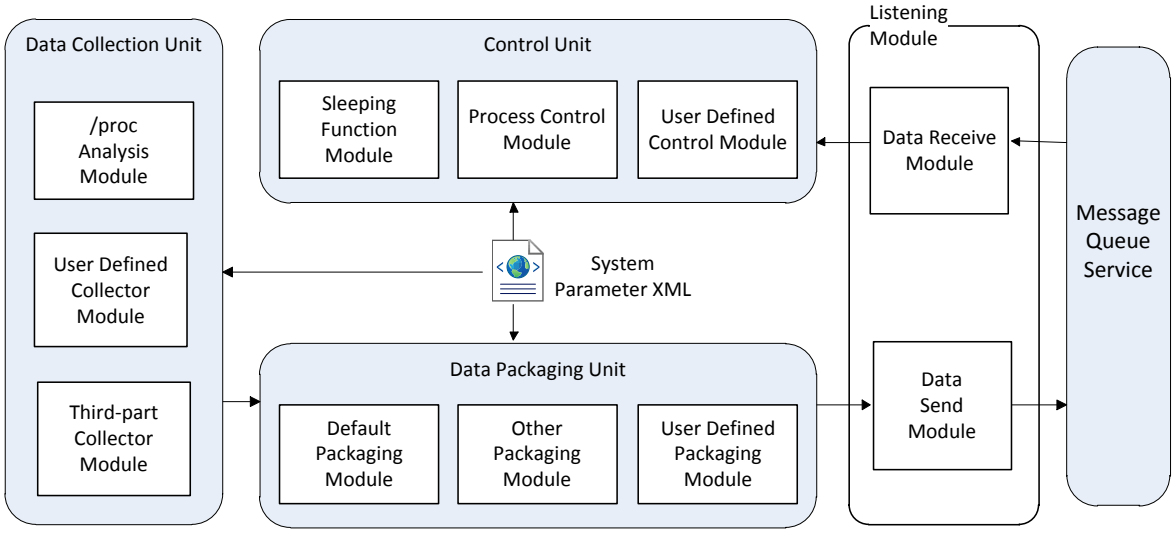

Figure 12. Architecture of Xmon Node Daemon.

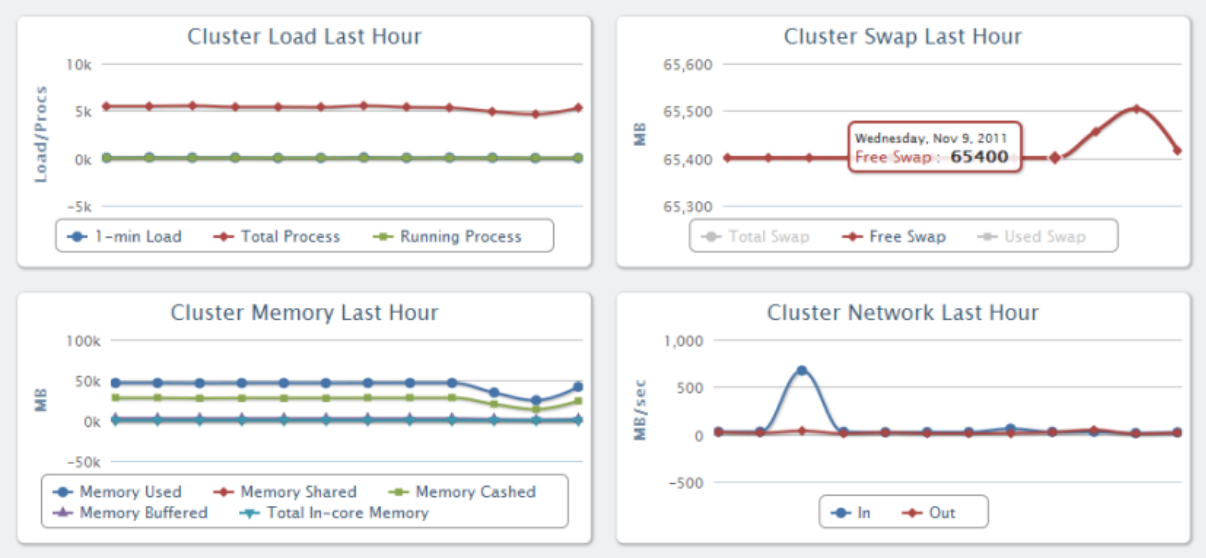

Figure 13. User Interface view.