Scalable Data Gathering for Real-time Monitoring Systems on Distributed Computing

Yoshikazu Kamoshida
The University of Tokyo
Hongo 7-3-1, Bunkyo-ku
Tokyo, Japan
kamo@logos.ic.i.u-tokyo.ac.jp

Kenjiro Taura
The University of Tokyo
Hongo 7-3-1, Bunkyo-ku
Tokyo, Japan
tau@logos.ic.i.u-tokyo.ac.jp

Abstract

Real-time monitoring is increasingly becoming important in various scenes of large scale, multi-site distributed/parallel computing, e.g., understanding behavior of systems, scheduling resources, and debugging applications. Dedicated networks on inter-site communications are rarely available for the monitoring purposes. Therefore, for real-time monitoring systems, reducing communication cost is important to handle a large number of nodes with limited network resources. We implemented a real-time Grid monitoring system called VGXP, with techniques for low cost data gathering. It tries to send only diffs to recent data, and adapts to the requested data freshness and tolerable errors to minimize required communication. We evaluate monitoring overheads of the proposed method on a distributed environment consisting of 8-sites with 500 nodes. In a realistic setting where the sampling interval is set to 0.5 seconds and the tolerable error to 2%, the CPU usage of the server to gather data from all nodes was 0.2% and the transfer rate was less than 5kbps. The transfer rate did not exceed 50kbps even if we gather a detailed per-process statistics.

1. Introduction

Quickly understanding the state of many computers is important to use a large-scale distributed computing environment efficiently. Real-time monitoring systems are particularly useful to grasp the overall load of the system, to avoid misuse of resources, and to understand the behavior of parallel/distributed software. Our goal is to make a large-scale distributed computing environment more accessible to non-expert users by providing a monitoring tool that can be used daily by them.

According to GMA [17], performance information, which is updated frequently, makes up the majority of the communication traffic of monitoring. Since the amount of monitoring data is proportional to the number of monitored objects and frequency of updates, it is very challenging to gather and to provide monitoring information of hundreds and thousands of computing nodes keeping the intrusiveness of monitoring low. When we monitor multiple sites, monitoring messages go through wide area networks. Reducing communication cost is much more important for such multi-site environments since network resources are more limited.

Existing monitoring systems such as PARMON[4], ClusterProbe [10] and Performance Co-Pilot[14] are based on the centralized daemon which gathers the information from the distributed agents running on every node. ClusterProbe employs a hierarchical architecture to enhance the scalability. A hierarchical architecture reduces the monitoring traffic by making a tree among monitoring agents. Agent processes on inner nodes of the tree filter and apply reductions to monitoring data. REED[3] presents the efficient methods of distributed join queries over static data tables in sensor networks. It reduces network traffic employing the techniques such as partial joins and pre-matching using Bloom filters. Instead of using the agents special to the monitoring system, PerfMC[11] uses SNMP to gather statistics from networked equipments. For inner-node informations, it runs the freely available SNMP agent provided by the NET-SNMP project[2]. Monitoring agents should be designed not to be intrusive to resources in the monitored system. Measurement of intrusiveness is usually done by amount of resources the agent used while monitoring. Intrusiveness is also measured by how much the application execution time is delayed[6].

Monitoring systems which provide web-based clients such as Ganglia[12] and Munin[18] are widely used because of its simplicity to users. These tools gather many kind of statistic information and show graphs as images using graph generation tools such as RRDTool[13]. Web-
based systems take charge of visualization on the server. Since generating graphs is not a lightweight task, these systems are not scalable against increase of clients. There are systems which uses data clustering techniques to achieve scalability of monitoring. To handle a very large number of clients, [5] dynamically classifies subscriptions to minimize the number of subscription checks.

The 3D Real Time Monitor [7] of GridPP monitors thousands of nodes spreading more than 300 sites joining in the EGEE Grid [1]. It provides the globe view using satellite imagery from NASA and displays running and scheduled jobs, job transfers and some of detailed information for each site on the Grid. It ensures that transfers are shown correctly nearly in the real-time order, but three minutes after they really occurred. The system has been developed for the purpose of monitoring job statuses, not machine statuses, so the available information is limited to them. A client requests 15k bytes of update information for every minutes as well as 10M bytes of an initial information. Initial information mainly includes a list of existing jobs, most of which is not used for visualization. It takes 1 ~ 3 minutes to start the client software because of the waiting time for initial informations.

There are some kinds of trends or patterns in data gathered by monitoring systems. We can predict or reduce errors in monitoring data using these characteristics. In the field of mobile systems, [9] estimates the network capacity with a pair of different exponentially-weighted moving average (EWMA) filters. It changes predicting algorithms depending on the stability of measured data.

In this paper, we describe a real-time monitoring system called VGXP, and propose methods to reduce the monitoring load/traffic for real-time monitoring systems. Our method reduces communication by carefully choosing data which make visible differences and sending only such data considering displaying capabilities and requirements of monitoring clients. Experiments show that the monitoring system which employs our method can monitor and show individual statuses of hundreds of nodes with a very low overhead. We retain information of all nodes without reducing them to an average or the summation. Since this method is applicable to the monitoring process of each node, it is possible to combine with the existing techniques for reducing data on the inner nodes of a tree of agents.

The rest of this paper is organized as follows. Section 2 gives an overview of VGXP, our lightweight monitoring system with a Java based graphical client. Then we give the detail of the proposed method to improve monitoring performance in Section 3. In Section 4, the experimental results are described, and finally this paper is concluded and future work are discussed in Section 5.

2. VGXP

The communication method presented in this paper is implemented as part of VGXP (Visual Grid Explorer) [8], a real-time monitoring system for multi-site distributed/parallel computing environments. Process management and monitoring on VGXP itself for fault-tolerance are done coordinating with GXP[16], scalable process manager. Configuration of VGXP is simple. Its minimum required settings are patterns of hostnames to be monitored, user names used to login with an SSH public-key authentication, and a port number of a server to communicate with a client. Rest of settings is automatically done by the system. The root privilege is not necessary to run VGXP. An impressive 3D visualization of the resource usage is available with a reasonably low load both on clients and monitored nodes. In addition, installing the client software is easily done from a web browser.

2.1. Process Architecture

VGXP consists of several kinds of processes, a client, a server, agents and event producers. A process diagram of VGXP is shown in Fig. 2. A client is a GUI application that visualizes monitoring data and interacts with a user. It connects to a server via network and receives monitoring data from the server. The server gathers the all required monitoring data and passes them to the client. The server does not directly receive them, but uses a hierarchical architecture. An agent process running on each monitored node receives monitoring data produced by an event producer process and sends them to the server through other agents on different
3. Performance Improvement on Data Gathering

It is very useful to find and to solve unknown problems on parallel applications if we can look over the whole distributed environment and check detailed information of nodes such as per-process statistics. However, communication cost of detailed information is high and we must moderate communication cost not to be intrusive to the environment. Therefore, reducing communication cost is necessary to utilize the distributed environment with monitoring systems.

In this section, we describe the method to reduce the cost of the communication for the monitoring data. The basic idea is to take advantage of the fact that a resource is often stable. A monitored resource is in a stable state if utilization of the resource per unit of time does not make interesting changes over time. For example, when a long-running CPU bound process is running, CPU usage is stays 100% and is thus stable. For another example, if the load average for recent 60 seconds keeps zero for a while, we say the load average for that period of time is stable. We reduce monitoring message by not sending data while the resource in the system is stable. For that purpose, we design the communication protocol of the monitoring agent that sends quadratic differential values to a client and does not send any data if their absolute values are small.

3.1. Sending Quadratic Differential Values

A typical sensor reports accumulated usage quantities of resources since it started its service. For the purpose of real-time monitoring, on the other hand, resource usages per unit time are rather important while accumulated values are often of little importance. When reporting for the $k$-th time is done at the time $t_k$ for each natural integer value $k$, resource usages per unit time are calculated as follows:
Given \( v_{n-1} \) as the accumulated usage of a resource at the time \( t_{n-1} \) and \( v_n \) as one at \( t_n \), resource usage per unit time for the time interval from \( t_{n-1} \) to \( t_n \) is formalized as follows.

\[
f_n = \frac{v_n - v_{n-1}}{t_n - t_{n-1}}
\]

Since a client requests for agents to report monitoring data periodically, we can assume \( t_n - t_{n-1} \) is always the same value \( T \) for every \( n \). A client can know \( \delta_n (= v_n - v_{n-1}) \).

Generally, polling or measurement is not strictly performed in the same interval. Therefore we adjust the data actually measured to reporting intervals. If we get the statistic value \( w_k \) at the time \( u_k \) and \( \delta \) for every \( n \), we assume \( \delta_n \). A client can visualize the resource usage information if it can know \( \delta_n(= v_n - v_{n-1}) \).

When the monitoring data of the resource is requested, the monitoring agent sends the initial value of the resource \((t_0)\) and the difference of resource usage rate multiplied by reporting interval \((\Delta_n = \delta_n - \delta_{n-1})\) periodically.

If \( v_n \) is a floating value, errors may be accumulated while the monitoring system is running. To avoid this, we periodically (but less frequent than sending monitoring data) send \( \delta_n \) or an absolute value to correct possible errors.

### 3.2. Reducing Communication Costs

When the resource in the system is in the stable state, rate of resource usage hardly changes. We can reduce the data traffic by refraining from sending data as long as the difference of resource usage does not change. An agent sends data only if \( \Delta_n \) is not zero. A client waiting for \( \Delta_n \) takes it as zero if each of the following is satisfied:

- The next data \((\Delta_{n+1})\) is received
- The timeout has expired

The timeout expires at \( t_n + D \), where the scheduled reporting time \( t_n \) and the delay to display data is \( D \). The above method can reduce the number of messages if \( \Delta_n \) becomes zero many times on each \( n \). Although the above method loses no information, we can achieve further reduction of messages if we accept the reasonable degree of information losses. Since the degree of information losses can be determined by the specification of visualization of a client, we call these types of optimizations “considering visualization requirements of a client.” Following three items are ones of the visualization requirements which are useful to reduce communication costs:

1. **Tolerable Error**
2. **Reporting Interval**
3. **Acceptable Delay**

We describe these three in order.

#### 3.2.1 Tolerable Error

Client software does not require exact values of all nodes because such data are not readable to a user. Reasonably designed client software has tolerable errors of values. For example, a client software which visualizes monitoring data as a bar graph, a change of the value is negligible if the effect of it is smaller than one pixel of the graph. Therefore, a client sends its tolerable error as the kind of monitoring data when it requests them.

A client sends \( \epsilon \) if the difference of resource usage per unit time less than \( \epsilon \) considered to be negligible for the client. An agent behaves as follows instead of the above method:

- If \(|\Delta_n + I_{n-1}| \leq \epsilon T\):
  - An agent does not send data and sets \( I_n = \Delta_n + I_{n-1} \).
- If \(|\Delta_n + I_{n-1}| > \epsilon T\):
  - An agent sends \( \Delta_n + I_{n-1} \) and sets \( I_n = 0 \).

\( I_n \) is the sum of currently ignored differences. This method loosens the condition to refrain from sending data. Therefore further reduction of messages is expected.

#### 3.2.2 Reporting Interval

Making a reporting interval longer can reduce messages more directly. This method is not very recommended because important changes might not be shown if the interval is too long. We believe reducing data considering tolerable errors is better because it will not miss important changes and it will be able to make more cut-down than simply prolonging the reporting interval if the state of the system does not change.

#### 3.2.3 Acceptable Delay

At the same time, we can combine multiple messages into one if we can know an message delay acceptable to a client. For the minimum reporting time of the unsent message of \( t_m \), the estimated maximum latency between an agent and a client of \( L \) and an acceptable delay of the client of \( D \), the maximum time until which the oldest message can be kept on the agent side is:

\[
t_m + D - L
\]

Therefore, messages can be sent every \( D-L \) seconds. Since the frequency that a message is produced is \( 1/T \), \( [(D-L)/T] \) messages can be buffered and sent at one time.
3.3. Implementation

A server, an agent and an event producer are written in perl and a client is a Java application. For communications among agents, we use TCP/IP sockets over SSH port forwarding so that they can communicate under the existence of firewalls. Agent processes make tree-structured network connections. VGXP has a self monitor which periodically checks increase and decrease of nodes to be monitored. When any changes of monitored nodes occur, firstly GXP logsins to new nodes using SSH and reconstructs a login tree. Then VGXP is restarted and given the same tree to make network connections. All of the other components communicate over TCP/IP sockets. Listening ports except for that of server listening for a client are dynamically determined to simplify configurations. An event producer scans a proc filesystem of Linux and reports system-wide statistics such as CPU usages, load averages and memory usages, and per-process informations such as command lines and CPU usage. For per-process data, we send a message if the resource usage of at least one process exceeds the tolerable error.

For visualization requirements, we calculate tolerable errors automatically according to the size of the window of the client and the angle of the 3D-bar graph. For example, if a client program provides a top view like Fig. 3, differences in resource usages can be largely cut off because resource usages are shown only by colors of the squares. The reporting interval and the acceptable delay can be changed by inputs from a user.

4. Experiments

We implemented the communication method proposed in Section 3 on our monitoring system VGXP, and evaluated its performance. We describe details below.

| Table 1. Machines used for experiments |
|-----------------|-----|-----|------------------|
| CPU      | GHz | K<sup>a</sup> | C<sup>b</sup> | Site ID(Num. of Nodes) |
| Core2 6400 | 2.13 | 2.6 | 2 | 1(58),2(14),3(30),6(33),7(14) |
| Pentium M  | 1.86 | 2.6 | 1 | 1(51),2(65) |
| Xeon 5110  | 1.60 | 2.6 | 4 | 8(1) |
| Xeon 5140  | 2.33 | 2.6 | 4 | 5(22) |
| Xeon 5160  | 3.00 | 2.6 | 4 | 1(1),2(1),3(1),5(1),6(1),7(1) |
| Xeon Prestonia | 2.80 | 2.6 | 2 | 4(69) |
| Xeon Prestonia | 2.40 | 2.4 | 2 | 4(64),8(38) |
| Xeon Prestonia | 2.80 | 2.4 | 2 | 4(37) |

<sup>a</sup> Kernel Release

<sup>b</sup> Number of CPU Cores

| Table 2. Elapsed time to scan a proc filesystem |
|-----------------|------|
| Num. of Processes | Elapsed Time(msec) |
| 63               | 6    |
| 117              | 11   |
| 212              | 22   |

(Measured on a Xeon Prestonia 2.40GHz node)

4.1. Experimental Environment

Experiments were done on 502 nodes of cluster computers spread to 8 sites in Japan. The number of nodes per cluster varies from 15 to 170, and all nodes are Linux. The kernel releases of 102 nodes are 2.4, and the others are 2.6. Details are shown in Tab. 1.

GXP, the process manager, copies program codes of an agent when it establishes login sessions to all nodes. In the experimental environment, it takes about 10 seconds to login and copy programs to all nodes. After the all agent processes start, they finish establishing required network connections and preparing to send monitoring data in about 20 seconds.

4.2. Load of Event Producer

The event producer used in the experiment scans a proc filesystem of Linux and reports system-wide and per-process statistics every half second. Although the interval of polling can be set longer value, we set the polling interval of the event producer fitting to the shortest reporting interval to evaluate communication cost when reporting intervals are different.

The time which takes to scan a proc filesystem mainly
depends on the number of processes in the node. Tab. 2 shows elapsed times to scan a proc filesystem in a node of 2.4GHz CPU. Scanning is done in short time if CPU is faster.

4.3. Effect of Message Reduction

Although our monitoring system processes monitoring data in real time, we used sets of log files which event producers generated for certain periods to estimate the effect of the message reduction. We feed logs from all nodes in an experimental environment to the program on which the data reduction techniques described in Section 3 are implemented. We calculated the number of messages and monitoring traffic under different values of parameters such as tolerable errors and reporting intervals. Analyses were done on the following two set of log files:

- idle log:
  1-hour logs when about 10% of nodes were busy (someone running jobs on the nodes)

- busy log:
  1-hour logs when about 50% of nodes were busy

Fig. 4 shows the number of monitoring messages under different values of tolerable errors. CPU utilization, the load average of recent 1 minute and per process information are shown for each set of logs. The black areas in the bottom of “process” bars indicate the numbers of messages of static information, which are always sent only once because they do not change during the process’ lifetime. The other areas indicate CPU utilization in the given update interval. The measurement has been done for each value of a tolerable error: 0%, 2%, 4% and 20%. The tolerable error of 0% means that sending data is refrained only if there are exactly no changes in the resource usage. The tolerable error of 2% means that ±2 percent of differences are regarded as no changes. The reporting interval is 500 milliseconds. The leftmost bar (ALL column) is the number of message if no message reduction technique is applied for reference. The figure shows that the number of messages for CPU usage is reduced more than 30% when tolerable error is zero, which causes no information loss. Messages are much more saved for data of the load average. This proves the effectiveness of sending quadratic differences. Because we send a message if the information of at least one process is changed, messages are not saved so much for data of per-process CPU utilization.

The effect of considering tolerable errors is impressive. More than 95% of messages are reduced by ignoring only ±2 percent of differences for the load average. Although per-process information cannot be easily reduced, more than 80% of messages are cut off for the busy set of logs.

The results of the both set of logs are not significantly different. Proposed message reduction technique is applicable to both of the busy-state systems and the idle-state systems.

Fig. 5 and Fig. 6 show amounts of data gathered to a server. These experiments were also done with the reporting interval of 500 milliseconds. The results of CPU usage and the load average have similar tendencies to that of number of messages. By considering tolerable errors, the data traffic becomes less than 5kbps for CPU usage and the load average. In Fig. 6, “name” groups represent the data of the process names, full command-line arguments and user names of owners. These do not change until termination of the processes. Therefore, the data rates of them are constant regardless of the degree of a tolerable error.

Fig. 7 and Fig. 8 show amounts of data gathered to a server under different reporting intervals (500, 1000, 2000 and 5000 milliseconds). Fig. 7 is the result when the tolerable error is zero. Fig. 8 is the result when the tolerable error is 2%. If the tolerable error is zero, the effect of reduction
is roughly reverse proportional to the reporting interval. On the other hand, if the tolerable error is 2%, changing reporting intervals can make little difference to the data rate. This is because considering tolerable errors sufficiently reduces the data and there is no place for further reduction of the data for the value whose update frequency is low.

4.4. Load of Gathering Data on the Server

Fig. 9 shows CPU load of gathering monitoring data on the server. The system-wide CPU usages were monitored during the measurement. The results of different tolerable errors are plotted. The series “ALL” means the case when no data reduction was performed. When monitoring 500 nodes, the CPU load was about 20% if tolerable errors are considered. The load of the server is lowered because considering tolerable errors causes the number of messages decrease.

5. Conclusion

A periodical communication is one of the most important characteristics of the data gathering for monitoring systems. Focusing on this, we proposed a data reduction method for communication by sending data only if the quadratic difference of value is large. We evaluated the effect of the proposed method by experiments on the environment of about 500 nodes spread out across 8 sites and found the case that our method can reduce more than 95% of messages and traffics compared to the method which sends data each time by ignoring only ±2% of changes in a setting where the sampling interval is set to 0.5 seconds. The communication traffic on a client is as small as several dozen kbps even if we gather a detailed per-process statistics.

Gathering and sending monitoring data will be delayed on a high-load system. We should investigate the effect of the proposed method to communication delays and study how possible we could integrate the proposed method with the method reducing per-node computational cost on gath-
ering monitoring data.

Realizing scalability for multiple clients is a challenging future work. Transfer of messages should be optimized considering different visualization requirements of clients. We are also working on refining the proposed method in this paper more applicable to different environments. It includes generalizing our system by separating it to OS specific event producers and platform independent data collecting and transfer as well as providing query APIs on collected data.

Acknowledgment

This research was supported by “Cyber Infrastructure for the Information-explosion Era” in the Japanese Ministry of Education, Culture, Sports, Science and Technology Grant-in-Aid for Scientific Research on Priority Areas.

References