CSCI B490 B534 Spring 2011

Project #3 Resource Monitoring System

1. Goal

To implement a system that monitors and visualizes the resource utilization in a distributed set of nodes using message broker middleware.

2. Introduction

Distributed resource monitoring is an important part of distributed systems. In large-scale computing environments, it is essential to understand the system behavior and resource utilization in order to manage the resources efficiently, to detect failures as well as to optimize the distributed application performance. There exist several distributed monitoring solutions, such as Ganglia [1] and Nagios [2], to apply into wide variety of distributed computing environments. Another example would be Amazon Cloudwatch [3], which allows the users to monitor the resource utilization of their EC2 cloud resources. While these systems need to be comprehensive, care must be taken to ensure that they don't disrupt or hamper the resources they are monitoring.

For this part of the project, you need to implement a system that monitors the CPU and memory utilization in a distributed set of nodes. The system should support the monitoring of bare metal as well as VM nodes running Linux OS's. Monitoring information needs to be collected and aggregated through the message broker and needs to be summarized to display the overall CPU and memory utilization percentages using graphs.

3. System architecture

There are three main components of this monitoring system: a **Message Broker**, **Monitoring Daemons running on nodes** and a **Monitoring UI**.

- **Message Broker:** a middleware that holds series of messages with specific topics, and waits for a Front-End Subscriber to pick the messages. i.e. NaradaBrokering[4], ActiveMQ[5], etc. AIs will setup instances of NaradaBrokering and ActiveMQ to be used by the students. Students are advised to prefix their topics with the group number (eg: G01_xyz) to avoid conflicts when sharing the same brokers.
    - **For NaradaBrokering, please use machine** "129.79.49.181", port number "3045".
    - **For ActiveMQ, please use machine** "129.79.49.181", port number "61616".

- **Monitoring Daemon:** a background process that runs on each compute node which captures and publishes the system resource utilization information (CPU and Memory utilization required) and other important usage information, to the Message Broker periodically. This daemon should

not interfere with the other running processes in the compute node. You need to implement this part.

- **Summarizer and Monitoring UI:** Summarizer should listen to the messages with a specific topic(s) from Message Broker and should summarize the collected information. These summarized information (overall CPU and Memory utilization) needs to be displayed using a cumulative graph of the targeted computing environment. The summarizer and the UI can be separate applications that communicate with each other or can be a single application. You need to implement this part.

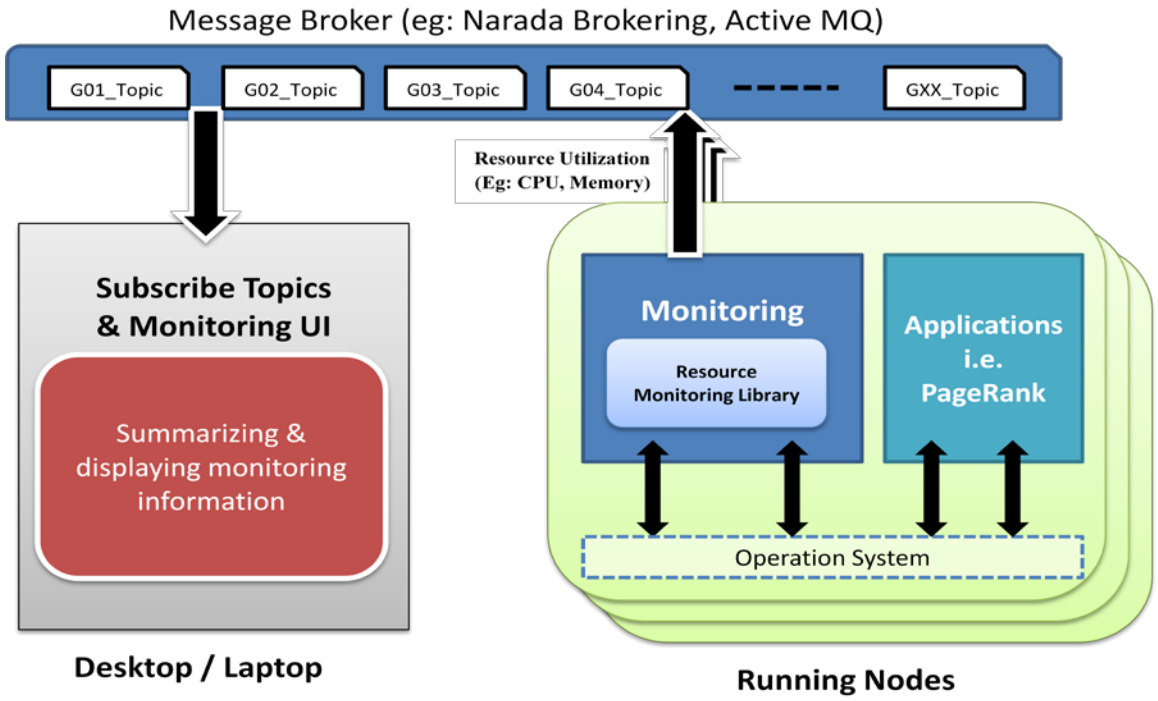

Fig 1. Overview Architecture

The following sections provide some instructions about implementing the **Monitoring Daemon and Front-End Subscriber**.

Monitoring Daemon

You can write the Monitoring Daemon in any language as long as your program provides the following functions:

1. Capture the system resource utilization information such as CPU and Memory utilization. If you are using Java, Sigar [6] can be used to capture that information.
2. Communicate and publish messages to the Message Broker. The message format must be "GXX_topic/….", where XX is your group number and topic can be any string.

Pseudocode

```
//Run this in a daemon thread
While(!done){
        perf = getSystemPerf();
        Message msg = new message();
        msg.NodeId = nodeID;
        msg.CPUPercentage = perf.AvgCPUUtilization;
        msg.MemoryPercentage = perf.MemUtilization;
        …….
        broker.publish(msg, topic)
}
```

The daemon should be able to run continuously utilizing very low CPU and memory. Make sure to avoid any memory leaks.

Monitoring Front-End.

This can be written in any language as long as your program provides the following functions:

1. Subscribe to the relevant topic in the Message Broker and receive the messages with monitoring information from the daemons running on the nodes.
2. Summarize the statistics from all the nodes. Calculate the overall CPU and Memory utilization percentages.
3. Draw real-time cumulative performance graphs with summarized resource utilization information. This must be GUI based. (No table or command-line text). If you use Java, JFreeChart [4] is a good option to draw the graphs. Figure 2 is an example UI developed using JFreeChart by SalsaHPC Team. The Blue area is the CPU usage, and the Green area is the memory usage.

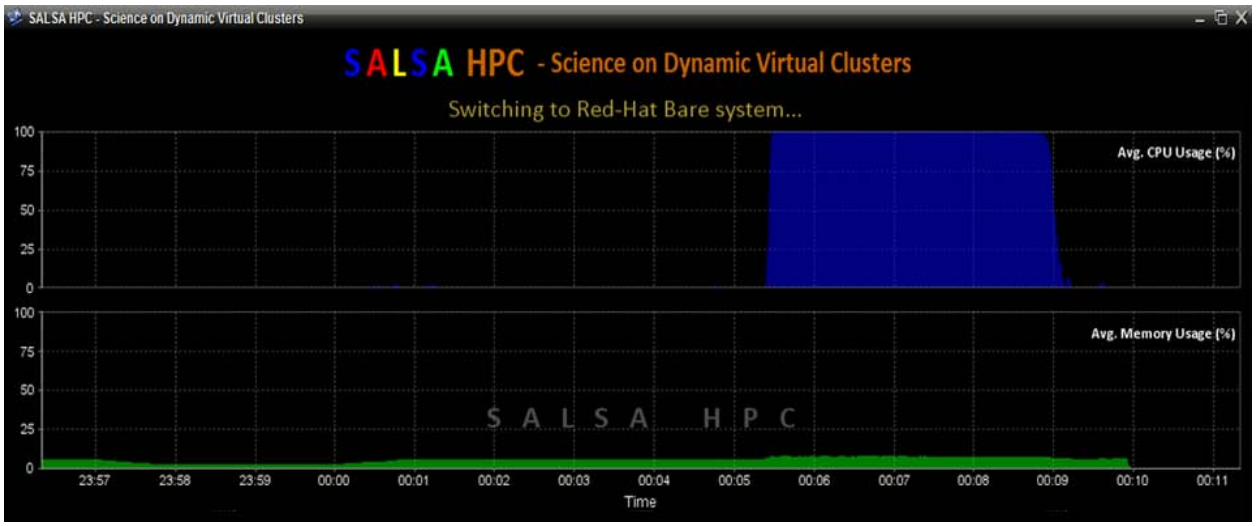

Fig 2. Real-Time cumulative performance interface with using JFreeChart

Pseudocode

```
Messages[] nodePerfMessages;
broker.subscribe(topic)
// this gets called by the broker when a new message arrives
onEvent(Message msg){
        nodePerfMessages[msg.NodeId] = msg;
}
UIUpdateThread{
        While(done){
                totalMem =0; totalCPU=0
                foreach(perfMsg  in nodePerfMessages){
                        totalCPU += perfMsg.CPUPercentage;
                        totalMem += perfMsg.MemPercentage;
                }
                updateGraph(totalCPU/numNodes, totalMem/numNodes)
                sleep (500)
        }
}
```

4. Deliverables (Due Apr. 13)

Each group is required to turn in the following items to OnCourse for this project with a filename of "B490_P3P1_GXX.zip" or "B534_P3P1_GXX.zip", where "XX" is the two digital number of your group. In addition, please complete your B534 project #3 webpage at same due time.

1) Source codes and executables of the Monitoring Daemon and Monitoring front-end.
2) Detailed instructions on how to configure and run each program should be given in README.txt. Try to keep the steps simple by using scripts for building and running the programs.
3) A document (.pdf) describing the design and implementation of the Monitoring system, in addition to screen captures of the UI with real cluster performance showed in the graph.
4) DEMO in class

References

[1] Ganglia, http://ganglia.sourceforge.net/
[2] Nagios, http://www.nagios.org/
[3] Amazon Cloudwatch http://aws.amazon.com/cloudwatch/
[4] NaradaBrokering, http://www.naradabrokering.org/
[5] ActiveMQ, http://activemq.apache.org/

[6] Sigar Resource monitoring API, http://www.hyperic.com/products/sigar http://sourceforge.net/projects/sigar/

[7] JFreeChart, http://www.jfree.org/jfreechart/

[8] B534/B490 Lecture 16 slides