Resource Monitoring System

Purshottam Vishwakarma
School of Informatics and Computing
Indiana University, Bloomington
pvishwak@indiana.edu

Bitan Saha
School of Informatics and Computing
Indiana University, Bloomington
bsaha@indiana.edu

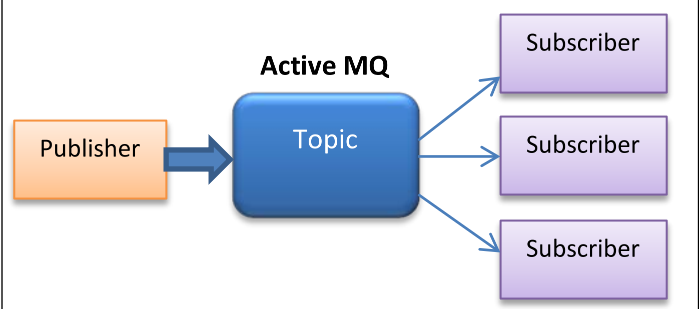
Introduction:

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 to apply into wide variety of distributed computing environments. 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.

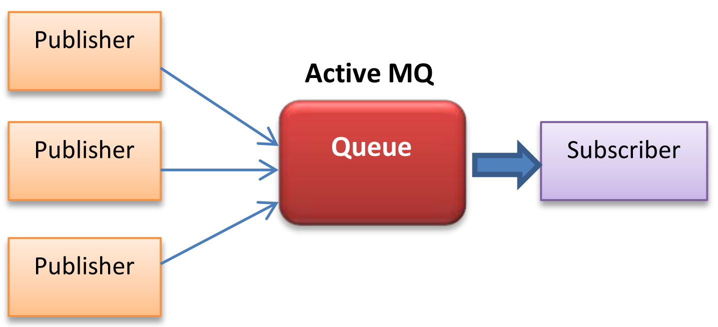
For this part of the project, you need to implement a system that monitors the CPU and memory utilization in a distributed set of nodes. The system should support the monitoring of bare metal as well as VM nodes running Linux OS's. Monitoring information needs to be collected and aggregated through the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

Techonology Used:

We have chosen "**Java**" as the programming language to implement both Monitoring system and the Daemon. We are using "**Active MQ**" in non-persistent mode as our Message broker.



Topics – Single Publisher and Multiple Subscribers



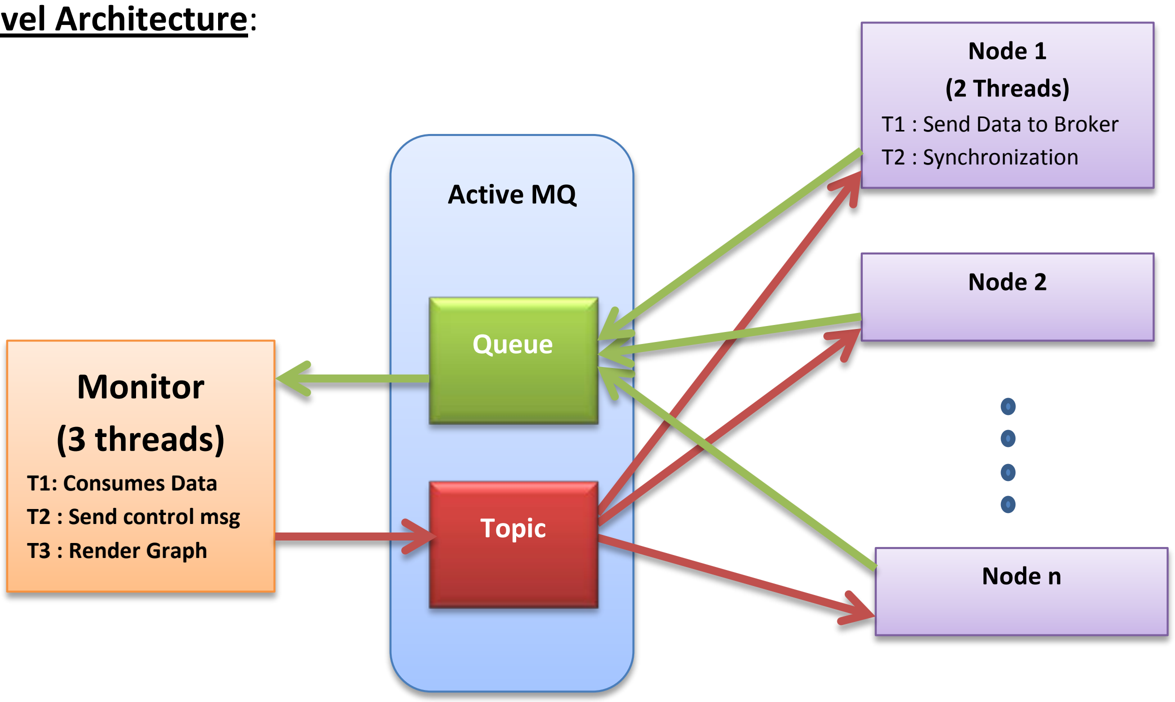
Queues – Multiple Publishers and Single Subscriber

We would be using both *topic* as well as *Queue* for communication in our implementation.

Topic is used for control messages (Synchronization) whereas *Queue* is used for transferring data and registration messages.

Our Monitoring system is designed to support n (n>= 1) number of nodes.

<u>High Level Architecture</u>:



→ **Control** Messages for **Synchronization**

→ **Data** Packets and **Registration** Messages

- Each node runs a daemon to collect CPU and memory Usage and routes it to the monitor through the Queue for visual representation.
- Monitor uses Topic to send control messages.
- New Nodes also use Queue to send registration messages to the Monitor.
- Monitor keeps a buffer for each node. So for n nodes it will have n buffers.
- One separate thread in the monitor reads all the buffers and renders the graph. After the data has been read it is cleared from the buffer.

<u>Registration & Synchronization</u>:

- Below is the data packet structure. Each member in the structure is described below:

<u>**Data Packet Structure**</u>
Message ID
State ID
CPU value
Memory value
Daemon ID

  - *Message ID*: This is an integer value used to identify the sequence number of the messages which is incremented with every data packet sent. The first data packet sent will have the message ID = 1 and subsequently incremented by 1.
  - *State ID*: This is used to identify the Phase of data transfer. The state changes every time a new node registers i.e. all the nodes will start sending data starting with message id 1 and this new state id.
  - *CPU Value*: This is the CPU usage at a given instance.
  - *Memory Value*: This is the Memory usage at a given instance.
  - *Daemon ID*: This is the combination of system name and IP address to identify each node uniquely.
- Whenever a new node joins the monitoring system, it sends a **registration** message to the Monitor through the Queue.
- Once the monitor receives the registration message it clears the buffer of each Node and sends a synchronization message with a new **State ID** to all the Nodes including the newly added node to *reset their Message ids to 1*.
- Hereon all the registered nodes will use this newly generated **State ID** along with message ID starting from 1 to send the data messages. This indicates a new phase of data transfer between Monitor and the Nodes. If the monitor receives a data packet with old state id, those data packets are discarded. This mechanism is used to synchronize all the nodes.

Auto Correction:

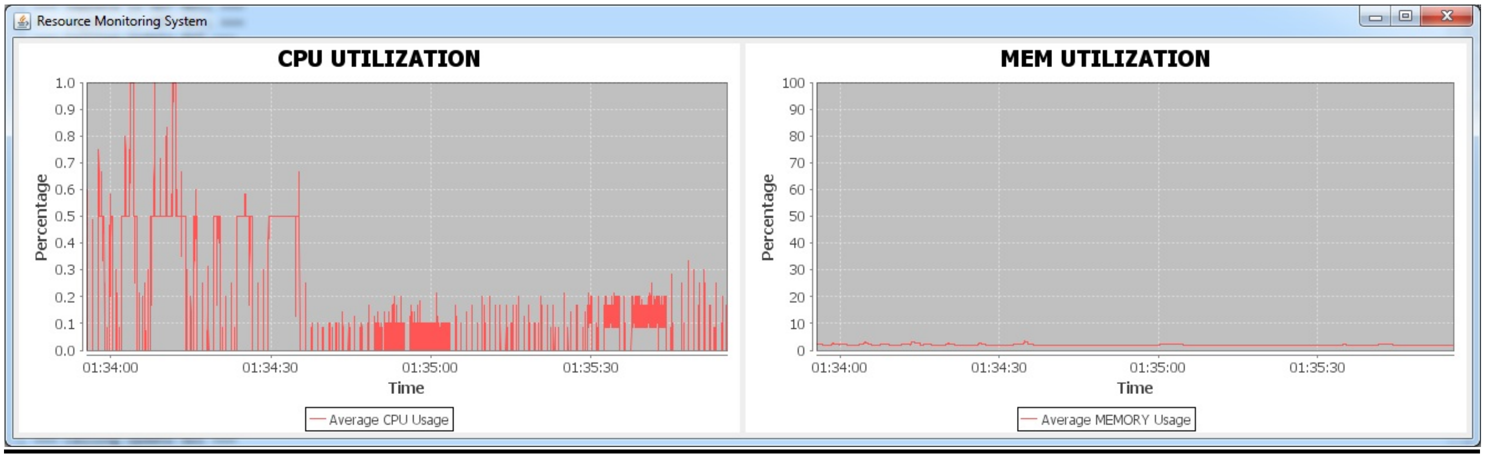
- We have incorporated *auto correction* feature in order to deal with the *lost data packets*. As an example, suppose there are 3 nodes in the system. Monitor will maintain 3 separate buffers for each node. The message received from each of the nodes are stored in the buffer corresponding to each node as shown the below diagram. Now suppose the monitor doesn't receive Message ids 999 and 1000 from Node 3. Note that we are using Active MQ with *non-persistent* mode. Hence requesting the broker to send the lost data packets is out of scope.

1005	1005	1005
1004	1004	1004
1003	1003	1003
1002	1002	1002
1001	1001	1001
~~1000~~	~~1000~~	Lost Packet
~~999~~	~~999~~	Lost Packet
998	998	998
997	997	997
Node 1 Buffer	**Node 2 Buffer**	**Node 3 Buffer**

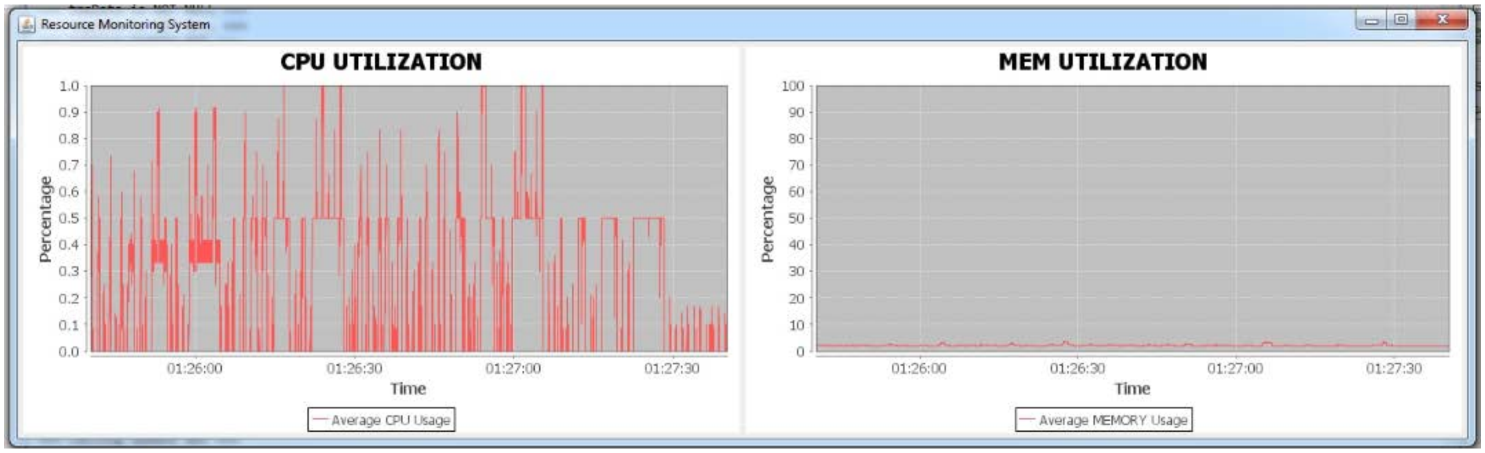
- In such a scenario, we will discard the message id 999 and 1000 from all other node buffers. This entails we display the data from message id **1001** after **998**.

Snapshots:

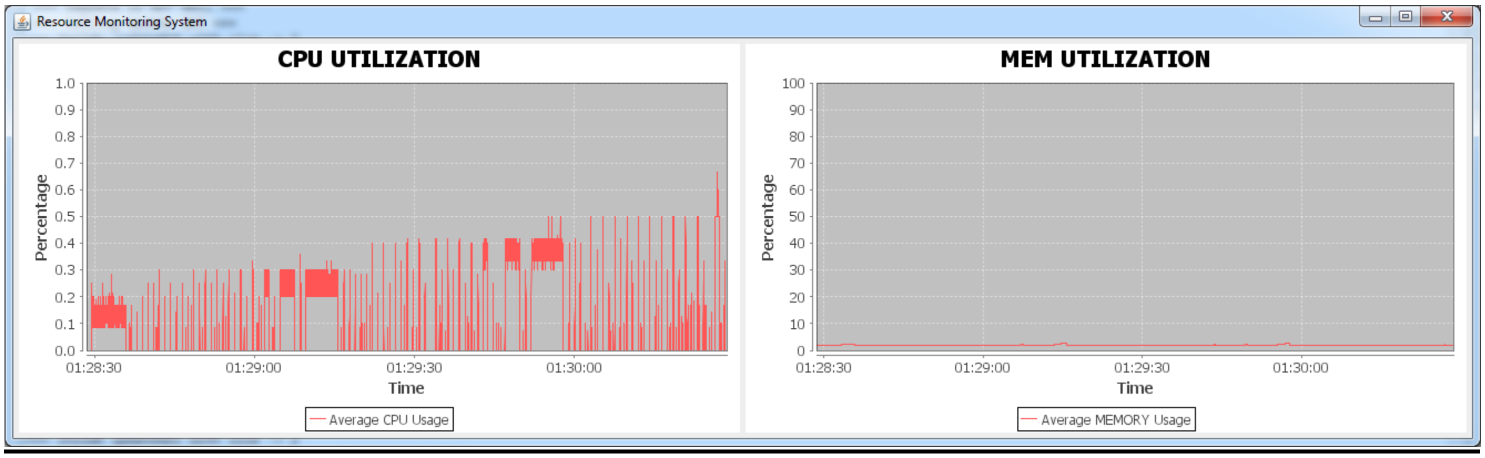
Snapshot 1: The below snapshot show the CPU usage when number of MPI process = 16 and 2 million URLS (between 01:34:00 and 01:34:30). After that the number of MPI processes decreases to 2 starting with number of URLs = 10K, 20K, 30K upto 2 Million. As the number of MPI processes increases the CPU utilization increases gradually. We tested this with number of MPI processes starting from 2, 4, 6, 8, 10, 12, 14, 16 with URLS 10K, 20K, 30K, 40K, 50K, 60K, 70K, 80K, 90K, 100K, 500K, 1M and 2M.



Snapshot 2: CPU Utilization and memory usage is maximum with number of MPI processes = 16 and 2 Million URLs.



Snapshot 3: CPU Utilization and memory usage gradually increase as the number of MPI processes increase along with number of URLs.



Future Work:

- We will enhance the autocorrect mechanism to support persistent Active MQ i.e. the monitor can request the broker to resend lost data packets and rebuild the buffer. In case the broker fails to deliver the missing packets the auto correct will perform similar actions as of now.
- We would also like to incorporate mechanism wherein other kind of control messages could be routed to the nodes. For e.g. providing a mechanism to control the daemon, change the data transfer frequency, etc.