Performance Analysis of sensors on SensorCloud

Santhosh Kumar Saminathan
Pervasive Technology Institute, Indiana University, Bloomington
sasamina@indiana.edu

Abstract

The objective of my independent study is to analyze the performance of sensors on sensor cloud in the pub-sub architecture by various factors namely time analysis, baseline analysis, middleware analysis, GPS sensor analysis and Hop level analysis.

Key Words: Sensor Cloud, Narada Brokering, FutureGrid, IaaS, Eucalyptus, Openstack Compute, Time Synchronization

1. Introduction

Anabas, Inc. and the Indiana University Pervasive Technology Institute have partnered to develop a Sensor-Centric Middleware System hereafter referred to as the Sensor Cloud.

The objective of the Sensor Cloud Project is to provide a general-purpose messaging system for sensor data called the Sensor Grid Server, and a robust Application API for developing new sensors and client applications. The key design objective of the Sensor Grid API is to create a simple integration interface for any third party application client or sensor to the Sensor Grid Server. This objective is accomplished by implementing the publish/subscribe design pattern, which allows for loosely coupled, reliable, scalable communication between distributed applications or systems.

This report consists of an overview of the sensor cloud architecture and a collection of test results that have been done, both past and present on the sensor cloud middleware. These tests are conducted are designed to find the limits of the middleware, the broker and the network in which these tests are performed.

2. Sensor Cloud Overview

The Sensor cloud implements the *publish/subscribe* design pattern to orchestrate communication between sensors and client applications, which form an inherently distributed system.

- Sensor Cloud Server creates *Publisher-Subscribe Channels* (Represented as a JMS Topic)
- Sensors acting as publishers create *TopicPublishers* to send messages to a Topic
- Client applications acting as subscribers create *TopicSubscribers* to receive messages on a topic
- Apache ActiveMQ is used as the default-underlying MOM and any other JMS style broker can be used as well.

Figure 2.1 shows a high-level overview of a typical deployment scenario for the Sensor Grid. The Grid Builder into logical domains deploys sensors; the data streams from these sensors are *published* as topics in the sensor grid to which client applications may *subscribe*.

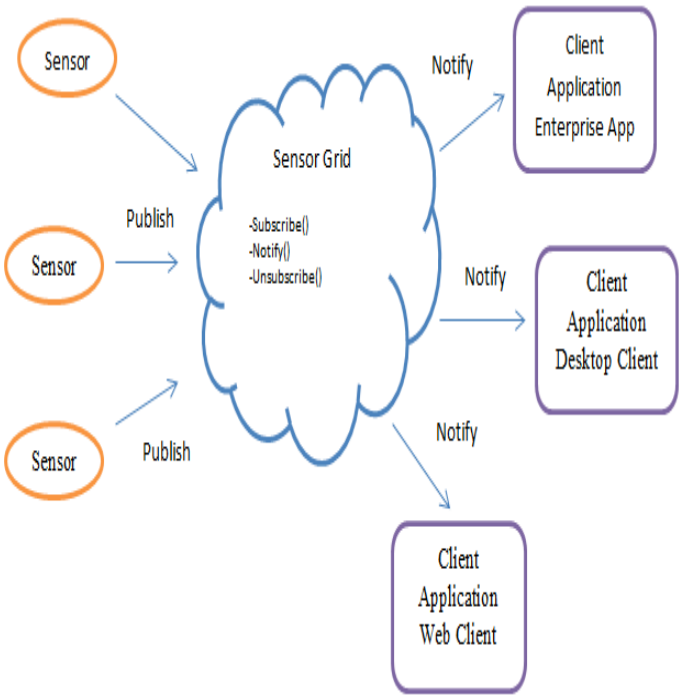

Fig 2.1

Examples of physical devices already implemented include

- Web/IP Cameras
- Wii Remotes
- Lego MindStorm NXT Robots
- Bluetooth GPS Devices
- RFID Readers

However Sensors can be made from chat clients, Power Point presentations, web pages virtually anything which produces data in a time-dependent stream can be implemented as a Sensor Grid sensor.

3. Large Scale Time Analysis

For our large-scale time analysis of sensors connected to the Sensor Cloud, we have a number of scenarios set up. We began with a series of time runs on a smaller scale of about ten medium sized instances on Futuregrid India Eucalyptus. We observed a trend in deploying these sensors.

3.1 Initial Small Scale Tests:

3.1.1 The Setup

Primarily, we made two runs. The first with a fixed transmission rate of 30 fps with varying sized data packets to simulate three video sensors of 320*240, 640*480 and 800*480. The second with a fixed data size denoting a video sensor with a resolution of 640*480 with varying transmission rates at 20, 30 and 60 frames per second. We observed the following sequence of latencies.

This was an initial attempt to get a feel of the way the system works. This effort was the first attempt made by us on this system. We set up instances manually on FutureGrid Eucalyptus. We attempted to test this setup on instances of varying sizes suspecting physical resources present in the node, however that did not seem a factor after a certain point as since after the point of optimum memory allocated to the domain, it was the network pipe which gave out.

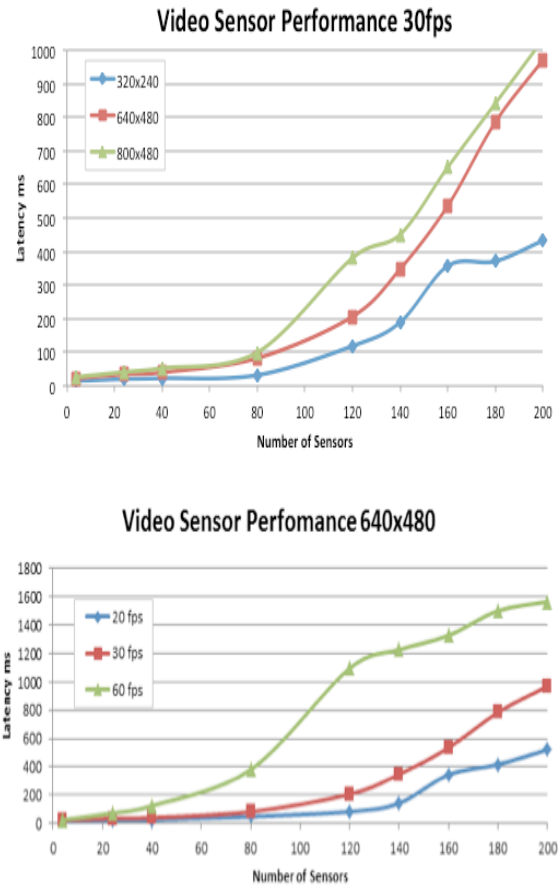

Fig. 3.1.1

3.1.2 The Observation

One common trend that we can observe in these graphs is that no matter what, the network becomes saturated more or less after we launch more than 80 sensors and get values from them at various publication rates. This attributes to the limited network pipe getting overwhelmed with our data packets. There is a steep increase after this number, which goes to show that as our number of sensors increase; the latency does too at an alarming rate thus reducing the overall reliability of our system.

Again, in case of the 2nd graph in Figure 3.1.1, we can see a much larger change in the plot as the frequency of publication increases thus causing over-queuing in the network pipes and increasing the latencies. This test was only able to convince us that we were able to reach a choking point due to network bandwidths over the FG pipes and this has nothing to do with middleware, as it was still capable of loading more sensors into it.

3.2 Larger Scale Tests:

3.2.1. The Pre-requisites

Before we could go on and launch a large number of sensors, there were a few issues we had to go through.

- The one of prime importance was setting up of a Distributed Brokering network. NaradaBrokering, which is primarily used in this software, is one such broker, which supports distribution over multiple nodes.
- Other than distributed brokering, there had to be certain configuration changes in the core Management files in the middleware to support hierarchical launch of the sub domains which can be intermediate ones or leaf nodes.

3.2.2. The Setup

The setup for this larger scale test was a little bit more complicated than the original procedure. It involved a lot of manual tasks of bringing up the domains and the brokers and the sensors, which was performed partly using scripts which log in and start the required services on dedicated nodes.

In our perspective, given all the resources we have at our disposal, we estimate this to be about 12000 sensors that we can have online at once. Our system can probably go larger given the resources but concerns over memory and usable IP's limit our scale to this extent.

These tests are performed on the FutureGrid Testbed. The FutureGrid project is a research effort used to provide resources to test out complex research challenges on cloud systems. We primarily use the 'India' node on the Futuregrid to run all our tests. India hosts Openstack compute and Eucalyptus now and these IaaS services are those that we use to create instances (nodes) to hold the sensors and the other middleware elements.

Common observations that we were able to note are shown below.

For 11250 sensors online, we initially subscribed to an increasing number of sensors and found that the latencies are as follows. These measurements are taken at 5-minute intervals.

Fig. 3.2.1 shows the latencies in milli-seconds on the y-axis, values taken every 5 minutes.

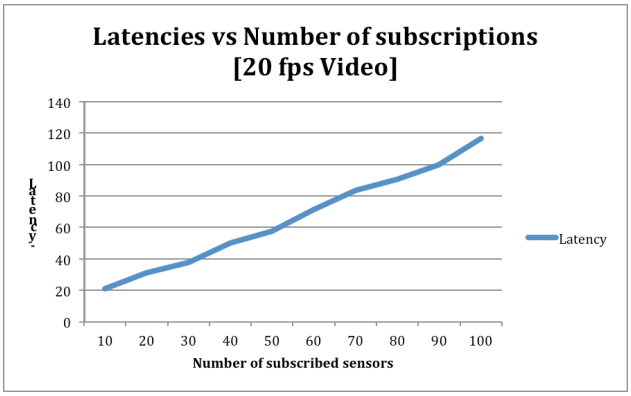

Fig. 3.2.1

Clearly we can see that the network suffers from some QOS when subscribed to sensors around a hundred. These are sensors publishing at 20 fps with a packet size of 7680B.

In our next round of tests, we subscribe to a smaller number of sensors but these publish at 30 fps with the same packet size of 7680B. The values again prove the system's capacity to hold upto 11250 sensors.

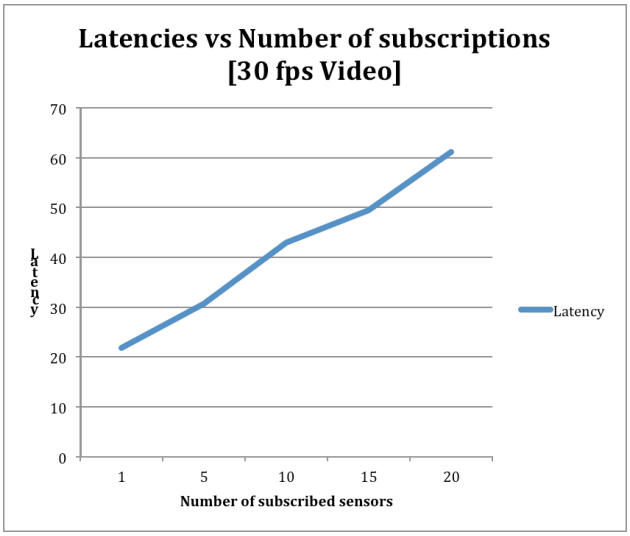

Fig. 3.2.2

In our final large scale run, we simulated a GPS sensor data using the benchmark sensor. This was done by making it send packets once every 2 seconds (rate of 0.5 FPS) with a size of 100B. This roughly denotes a standard GPS sensor packet. We subscribed to 200 of these and observed latencies.

Again, supporting upto 200 GPS sensors is well within the system and the network pipes. However, issues will arise when we support more than a thousand GPS sensors, as we will discuss later.

4. Baseline Analysis

The baseline analysis tests that we planned to make on the Futuregrid were already done by Alex Ho and his team of Anabas Inc. We will go through the outline of the tests they made in this section.

The primary goal of this baseline analysis was to observe network level throughput of the Futuregrid systems. This will give us an idea of the time values that we calculate are true values or have some resting on internal communications within the Futuregrid.

The first test, which was run, was to measure the bi-directional throughput value of the nodes on the FG network. This is done by acquiring a couple of instances on all the nodes Hotel, India, Sierra and Foxtrot using the ping command together with the iperf command for measuring packet loss and round-trip latency under loaded and unloaded network between all 2-combination of the set of four clouds selected. We can see their results on the combinations of the connections of the nodes.

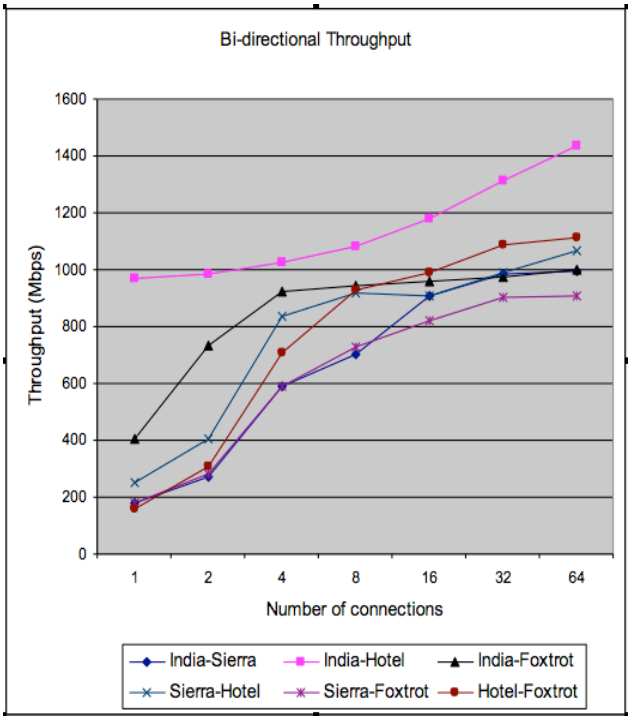

Fig. 4.1

While the maximum bi-directional throughput between any 2-combination ranges from 900 Mbps (on Sierra/Foxtrot pair) to 1,400 Mbps (on India/Hotel pair), we find the total iperf throughput in FutureGrid is over 800 Mbps when we connect any pair of cloud instances on distinct clouds with more than 16 connections in each direction.

We use the ping tool to measure network latency and packet loss between two clouds. Figure 4 shows the throughput between any 2 clouds in our experiments either levels off or starts to level off at 32 iperf connections for all but the connection between India and Hotel.

Results show ping packet loss rates in unloaded network for all the 2-combination of clouds were 0%; while the highest ping packet loss rate is 0.67% between the India/Hotel pair. The results indicate a highly reliable FutureGrid network under the experimental conditions.

Instance Pair	Unloaded Packet Loss Rate	Loaded Packet Loss Rate
India-Sierra	0%	0.33%
India-Hotel	0%	0.67%
India-Foxtrot	0%	0%
Sierra-Hotel	0%	0.33%
Sierra-Foxtrot	0%	0%
Hotel-Foxtrot	0%	0.33%

For baseline information we measure ping round-trip latency between 2 cloud instances on Sierra for the unloaded case and loaded cases with 16 and 32 connections before we conduct the same experiment on distributed clouds. We find latencies for the unloaded and the two loaded cases between two virtual machines communicating on the same cloud no higher than 1.18 milliseconds. Thus, we could reasonably assume for the ping experiments on distributed clouds the measured round-trip latencies are mainly due to distance between clouds. Virtual machine overhead is negligible in these experiments.

Ping round-trip latency for all six combinations of pairs of clouds is measured. We find the lowest average round-trip latency of about 18 milliseconds between India and Hotel in a loaded condition (see Figure 5). India and Hotel has the shortest distance between any 2 of the four clouds; and thus, is expected to show the lowest round-trip latency here.

We observe the highest ping round-trip latency in a loaded network condition is about 145 milliseconds on the Sierra and Foxtrot connection (see Figure 6). Although the inter-cloud latency between Sierra and Foxtrot is the highest due to its longest distance between any two of the

four selected clouds, we note that a round-trip latency below 300 milliseconds still meets a requirement for acceptable quality of service for collaboration applications with stringent network requirement like that of VoIP.

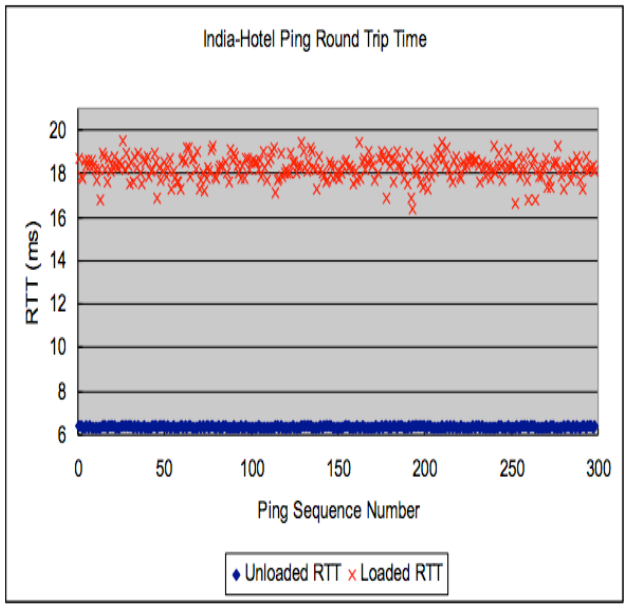

Fig. 4.2

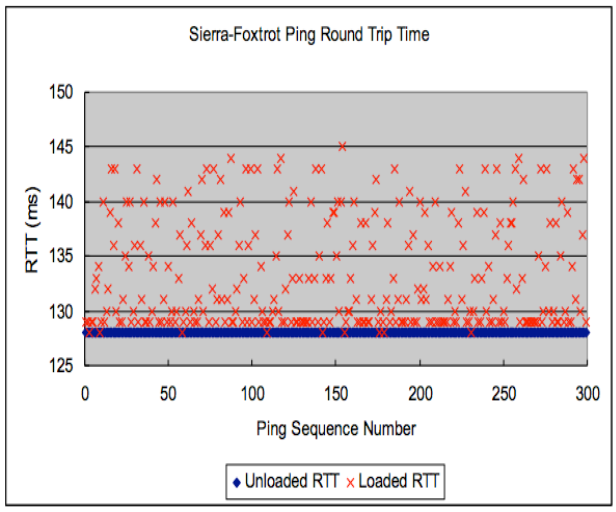

Fig. 4.3

limited initial results indicate that FutureGrid can sustain at least near 1 Gbps inter-cloud throughput and is a reliable network with low packet loss rate.

Credit to Alex Ho and his team for these measurements.

5. Middleware Analysis

5.1. Key Areas

In the middleware analysis section, we focus primarily on the points of strain on our middleware. There are a few areas of our system that are likely to fail here. Primarily this can be the broker.

We were able to observe the case where when we have a large number of GPS sensors (by large, in this scope I am talking about a few thousands) we are able to see that when we run the client application to see the output of the GPS sensors, we can see messages intimating us of a possible connection loss within one of the brokers due to a communication overload.

Another key area likely to fail is the middleware system itself. Although, we have not approached this magical number, which causes our middleware system to crash, I know that we have not yet broke into its inherent capacity to handle the load. Once we do hit this value, the core functionality of subscribing to topics that is the promise of the middleware system.

Another area where failure might be a possibility is certain natural causes of network or node failure. Although there exists a single point of failure to the root domain of the entire setup, Futuregrid is inherently a highly reliable system.

One last point of failure in our middleware system is the hardware resources available, mainly the memory that is used by the container service to allocate worker threads to the execution of many sensors. Any such cases are best sorted out beforehand and we can decide on the amount of memory to allocate to the container service and the amount of sensors.

5.2. Time Syncing

When we are faced with the problem of time syncing, there are many possible solutions we can turn to. One possible way to go about facing this problem is to make changes in our middleware to calculate the latencies of incoming packets by contacting the server for the current timestamp and then using the current timestamp and the timestamp on the received packet to figure out the latency. A rough representation of how that works is as follows.

There is an inherent overhead in making a call to the external node to calculate the most recent timestamp but within a single datacenter, these minor timekeeping changes can be ignored.

Another possible method of timekeeping is to use an external software, which automatically syncs up the current system it is installed in with an external server. One way to go about this is to have a common pool of servers from which we choose from to keep the nodes synced up. This is the approach we use since it involves little code change. One problem with this approach is that this method requires the node to be up for a certain amount of time before it synchronizes all the other nodes to this time. The software that we use for this purpose is called 'Chrony' and we include a pool of servers to which we synchronize to in its config file and wait for a particular window of time for which a system has to stay up.

The figure 5.2.1 shows the set up in case we decide to use the timekeeping system maintained by the datacenter.

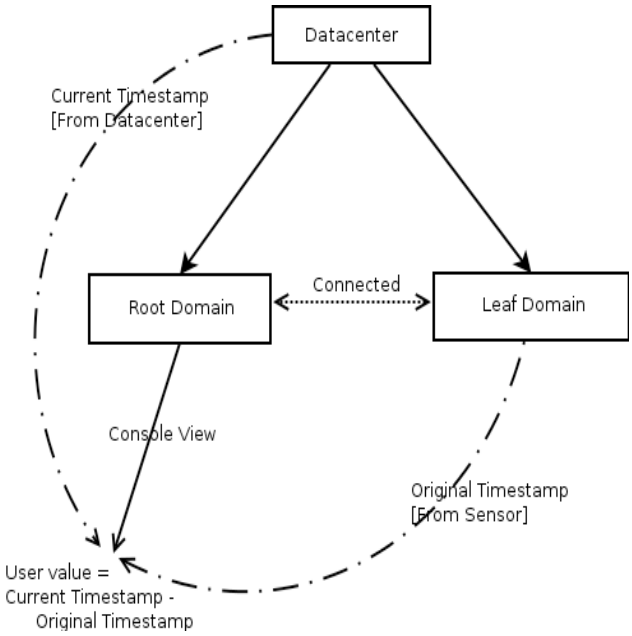

Fig. 5.2.1

6. Single Domain Level Analysis

When we were testing on a much smaller scale on a single domain, there were quite a few interesting results that we observed.

Initially, there is an inherent limit within the domain, which limits the NaradaBroker to not hold more than roughly 580 connections [sensors]. Connecting domains to multiple brokers can increase this number. Which are connected with each other in the form of a broker pool. Creating such a broker pool to which we can connect many domains and launch many sensors on those domains, a lot past the standard limits.

Creating such a broker pool requires distributing the brokers across the network in such a manner that each broker present is connected to every other broker present. Initially, every broker, which is brought up, connects to the root broker first and gets its assigned logical address. It then has to connect to every other broker connected to the root so that the topics to which the data is published is available to every broker on the network. When we bring up a domain, we connect this domain to every broker in the broker pool. This is the method we use to get up a large number of sensors [of the order of 12000].

7. GPS Sensor Analysis

Testing scenarios with the GPS sensor is slightly different from those used to test the benchmark sensors. This is due to the fact that once these sensors are launched, they immediately start publishing data and this data flows through the connected root domain.

This caused an interesting test scenario for us where we were able to get large numbers of GPS sensors online at a time and they were live [around the order of 3000]. However, from when this count value crosses a thousand, we will be unable to view the data published from these sensors as the broker will tend to get overworked and that results in a loss of connection on the client side.

8. Narada Broker Scalability Tests

A series of scalability tests were also done on the NB level using the middleware. These tests were to verify the work done by Alex Ho in the past to test the number of clients that one can scale with increasing number of brokers.

Alex had confirmed earlier that a single Narada broker can be used to support around 2000 clients comfortably for a simple low end video sensor. When we use the terms low end or high end video sensor we can assume the following. The low end video sensors are simulated sensors publishing at the rate of 10 fps a data packet of 1024 bytes and a high end sensor is one simulated by

publishing at a rate of 33 fps a data packet of size 7680 bytes. The tests that Alex performed with a low end sensor was confirmed again by us and repeated with a higher end video sensor.

The observations that we noted are shown below. We can find that Alex's observation of a single broker supporting only 2000 clients was confirmed as the latencies grew to become unbearable after that. But when these clients were distributed over multiple narada brokers, we could notice that the latencies were very well within the tolerated range and so we can say for sure that as we increase the number of brokers, we can scale out the number of clients as well. This observation was noticed from the figure 8.1.

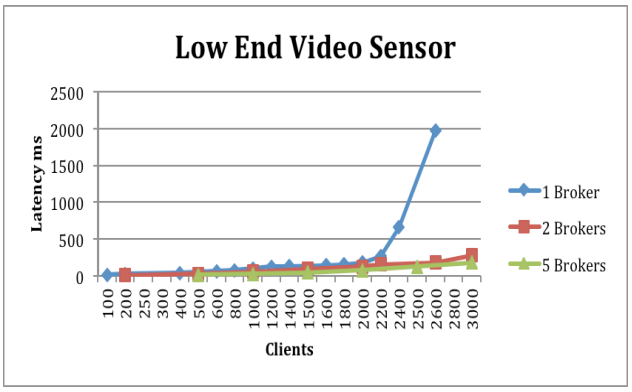

Figure 8.1

A similar set of tests was also replicated for the case of using a high end video sensor. We noticed a trend similar to the earlier tests but we were able to support way less clients than we did for the Low end video. This could be any of the two possible reasons. The first one being the limiting factor in a single Narada Broker to handle only that much data and the broker giving out. The other being the network pipes being used in the tests. In our tests, we use a standard 1 Gbps pipe for communication and so after a number of clients, this pipe has a greater change of getting filled and that causing the noted latency abnormalities. These trends were similar when we increased the number of broker units. We were able to scale the number of clients subscribing to this high end video sensor but it was nowhere close to the number of clients that a low end client could subscribe. These observations were drawn from the following performance chart with regard to the high end video sensor.

Again, in theory, the Narada broker should be limited only by the number of messaged per second and not the size of the data packet in the high end video sensor so technically, it must be able to support 1/3rd of the clients that a low end sensor supports. But this assumption cannot be measured with the network configuration as we use only a 1 Gbps pipe. Absolute confirmation can be obtained by repeating similar tests on an Infiniband connection so that we are not network stymied. These tests will be completed later.

The tests completed for varying number of brokers and clients to support a high end video sensor on a 1 Gbps connection can be found in Figure 8.2.

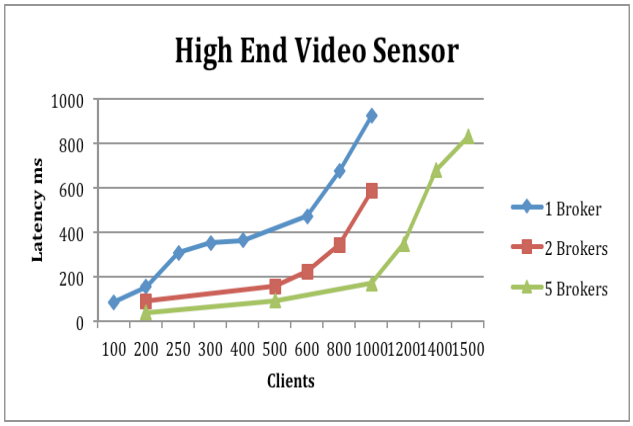

Figure 8.2

9. Geometric Tests

After verifying scalability tests with the video sensors, we repeat a single test on multiple nodes spread out geographically all over the country and the latencies are verified. The test repeated here is the latency test on a low end video sensor with it supporting clients over two narada broker nodes. This test was carried out with the clients spread out over the Sierra node on Futuregrid located in San Diego and the Hotel node located in Chicago. We were able to note significant lags in the latencies due to the distances of the nodes.

The latencies observed with the Sierra node was greater than that observed from Hotel as the Hotel node is closer to the local India node and the Sierra node is across the country. Results observed are plotted in the Figure 8.3.

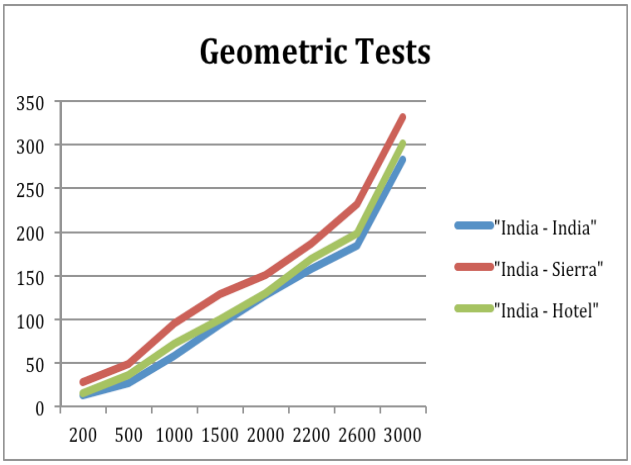

Figure 8.3

10. Summary

On the whole, a whole series of tests had been made and are still being planned. From the results that we have, I think its fair to say that this middleware system is robust in its own way. It supports scaling of sensors to an extent once we figure out how it can be done.

Performance metrics taken so far do show us that we have come across a number of scenarios on the large scale where the scalability of the middleware has not been brought into question since we seem to be running out of resources well before then.

On smaller scale analysis, we see a number of areas that had issues and have been worked with including scaling the brokers to support a larger sensor base and so on.
To conclude, I would like to thank Dr. Fox for providing me a chance to work on this system and Ryan Hartman for helping me along.

11. Acknowledgements

This project uses the FutureGrid testbed. The FutureGrid a project supported by the National Science Foundation under Grant No. 0910812 to Indiana University for ”FutureGrid: An Experimental, High-Performance Grid Test-bed. I would like to thank Dr. Geoffrey Fox for giving me an opportunity to work on this project. I would like to thank Ryan D Hartman for the constant support and guidance. Also I would like to thank Sharif Islam for the support via mail.

12. References

1. NaradaBrokering. Scalable Publish Subscribe System http://www.naradabrokering.org/documents.htm

2. Geoffrey Fox. FutureGrid Platform FGPlatform: http://www.futuregrid.org.

3. Pallickara, S. and G. Fox, NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids, in ACM/IFIP/USENIX 2003 International Conference on Middleware. 2003, Springer-Verlag New York, Inc. Rio de Janeiro, Brazil.

4. Eucalyptus Open Source Cloud Software. Available from: http://open.eucalyptus.com/

5. Geoffrey C. Fox, Alex Ho, Eddy Chan, Measured Characteristics of FutureGrid Clouds for Scalable Collaborative Sensor-Centric Grid Applications

6. Experiences with Eucalyptus: Deploying an Open Source Cloud [Rick Bradshaw, Argonne National Laboratory, bradshaw@mcs.anl.gov, Piotr T Zbiegiel, Argonne National Laboratory, pzbiegiel@anl.gov]