Architecture, Performance, and Scalability of a Real-Time Global Positioning System Data Grid

Galip Aydin[1,2,*], Zhigang Qi[1,2], Marlon E. Pierce[1,**], and Geoffrey C. Fox[1,2]
[1]*Community Grids Lab*
[2]*Department of Computer Science*
Indiana University
501 North Morton Street
Suite 224
Bloomington, IN 47404
Ph: 812-856-1212
{gaydin, zqi ,mpierce, gcf}@indiana.edu
[*]*Corresponding author (primary)*
[**]*Corresponding author (secondary)*

Yehuda Bock
Cecil H. and Ida M. Green Institute of Geophysics and Planetary Physics
Scripps Institution of Oceanography
La Jolla, CA 92093
ybock@ucsd.edu

Abstract: We describe the architecture of our real time SensorGrid system, which supports the real-time message routing and processing of Global Positioning System data. The current system processes 1 Hz position data from 85 stations in southern and central California. Our system is built on a computer network-enabled, topic-based publish/subscribe system and is extensible to other data sources. In order to determine the performance and the upper limits of scalability, we have conducted and present here a set of systematic test evaluations of our implementation. Through these tests, we are able to show that performance does not degrade over time, that the system will handle up to 1000 simultaneous data providers or data consumers with a single message broker, and that multiple message brokers can be linked to provide scalability beyond 1000 providers or consumers.

Keywords: Real-time GPS, message-oriented middleware, publish/subscribe middleware, Grid computing

1. Introduction

Real time sensors are changing the way we acquire data about our environment. Recent advancements in sensor technologies such as micro-circuitry, nano-technology and low-power

electronics allow sensors to be deployed in a wide variety of environments [1-6]. Environmental monitoring, air pollution and water quality measurements, detection of the seismic events and understanding the long-term motions of the Earth crust are only a few areas where the extent of the deployment of sensor networks can easily be seen. Extensive use of sensing devices and deployment of the networks of sensors that can communicate with each other to achieve a larger sensing task will fundamentally change information gathering and processing [7].

Real-time GPS data sources have the capacity to produce tremendous quantities of data, which might be more than the traditional systems can handle in normal operation. For instance the Southern California Integrated GPS Network (SCIGN) [8] has deployed 250 continuously-operating GPS (CGPS) stations in Southern California, and EarthScope's Plate Boundary Observatory (PBO) (http://www.earthscope.org/), is installing hundreds more stations in the Western U.S . The GPS Earth Observation Network System or GEONET in Japan consists of an array of 1200 GPS stations that cover the entire country with an average spacing of about 20 km [9]. For example, the Scripps Orbit and Permanent Array Center (SOPAC) is in the process of upgrading SCIGN and PBO CGPS stations to real-time operations as part of the California Real Time Network (CRTN). Data are collected once per second and stations positions are computed on-the-fly with a latency of less than a second. These networks are capable of producing terabytes of measurements per year.

Table 1 shows approximate amount of data produced by CRTN stations. The observations obtained initially from a proxy server are encoded in an open binary format called RYO. The table shows the increase in size for different encodings of the same observations.

Table 1 – Approximate amount of real time data produced by CRTN. The stations produce RYO binary data, which can be translated to ASCII and the Geography Markup Language (GML).

		Message Format		
	Time	RYO	ASCII	GML
CRTN Site Positions (9 Stations)	1 second	1.5KB	4.03KB	48.7KB
	1 hour	5.31MB	14.18MB	171.31MB
	1 day	127.44MB	340.38MB	4.01GB
	1 month	3.8GB	9.97GB	123.3GB
	1 year	45.8GB	119.67GB	1.41TB
Entire SCIGN Network (250 stations)	1year	1.23TB	16.18TB	160TB

The rapid proliferation of sensors presents unique challenges that differ from traditional computer network problems. Several studies have discussed the technological aspects of the challenges with the sensor devices, such as power consumption, wireless communication problems, autonomous operation, adaptability to the environmental conditions and load balancing [10] [11] [1]. We address a different problem domain in our work: we describe the architecture and implementation of a scalable messaging system for processing and delivering

real-time GPS position data streams to end consumers, which may be applications or (through Web interfaces) humans. We believe our architecture is general and can be applied to other types of data.

Scientific applications that require processing of huge data sets are increasing in number with the evolution of computing resources, network bandwidth, and storage capabilities. At the same time some of the applications are being designed to run on real-time data to provide near-real time results. Such applications are gaining ground in systems like Crisis Management [12] or Early Warning Systems [13] [14] because they allow authorities to take action on time. Earthquake data assimilation tools are good examples of this group since they use data from Seismic or GPS sensors. However, most of these tools currently consume data from repositories and they do not have access to real-time data due to several reasons.

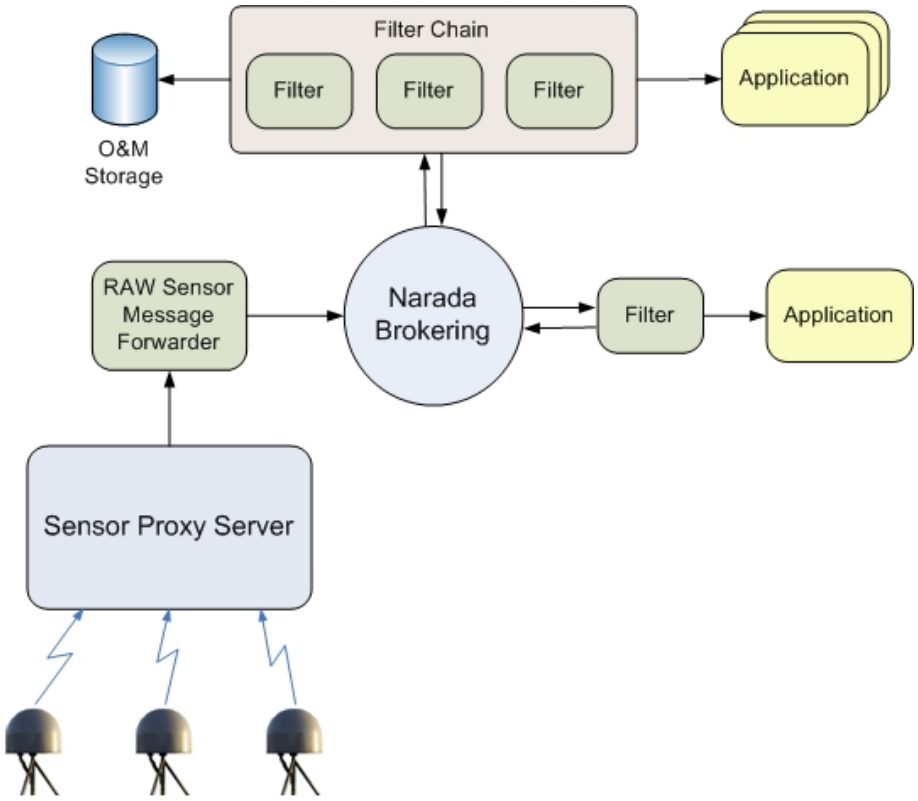

Figure 1 - Overall SensorGrid architecture. Station positions from persistent GPS networks (bottom left) are delivered to a proxy server and can be obtained by TCP/IP connections. These raw messages are ingested by the message forwarder, which publishes data streams to a message broker (using NaradaBrokering software; see text). These messages can be processed by various network-enabled message filters, which act as both publishers (message sources) and subscribers (message sinks).

Figure 1 shows the overall SensorGrid architecture, which contains several filters for processing, converting or aggregating data streams. We use the NaradaBrokering [21] messaging system for over-the-wire message transfer. NaradaBrokering can also be configured to provide additional qualities of service such as security [22] and reliable delivery [23]. This approach allows us to reformat messages on the fly, so we can convert (for example) the original

RYO formatted source data into Open Geospatial Consortium's Observations and Measurements (O&M) format, while preserving the original message. Subscribers can connect to the filter chain at any point to get the messages for the desired stations in the desired format. The architecture also can be extended (in the Object Oriented programming sense, using inheritance from base Filter classes) to incorporate more sophisticated filters, such as data analysis and event detection applications. We describe this work in more detail in [24].

A key concept in our system is the publication and subscription of messages on topics. As we will describe, "messages" in our system are data packages combined with header information that can be sent over network connections. A topic-based publish/subscribe system is an example of a many-to-many network communication system in which publishers send messages to message brokers on name-specified topic. These messages are then routed to all topic subscribers. Thus the publishers and subscribers on a particular topic do not make direct network connections to each other, as in the case of client-server systems such as Web browsers and servers. As we will discuss, this approach allows us to build up complicated applications using chains of publishers and subscribers that are individually responsible for relatively simple, atomic tasks. Publishers and subscribers can connect to the mediating message broker using a number of different network protocols, although we use TCP/IP in the system implementation described here.

The remainder of this paper is organized as follows. Section 2 describes the Scripps Orbit and Permanent Array Center's real-time GPS network. We also describe our system architecture in more detail. Section 3 provides a detailed description of our system tests for performance and scalability. Section 4 provides a paper summary and acknowledgments.

2. Real time Data Grid Implementation for Global Positioning System Networks

The Scripps Orbit and Permanent Array Center (SOPAC) maintains and operates the California Real Time Network consisting of distributed GPS sub-networks transmitting publicly available 1 Hz station positions with a latency of less than 1 second. . Raw data from the GPS stations are collected by the Geodetics RTD Pro/CLP software suite and instantaneous station positions are computed on-the-fly. In this section we describe the implementation of the aforementioned technologies.

Figure 2 displays real-time GPS subnetworks in Southern California. The triangles represent the continuous GPS stations (i.e., stations not currently accessible in real-time) while the blue and red circles represent the real-time stations (i.e., the station's data can be accessed over the network as it is measured by the sensor).

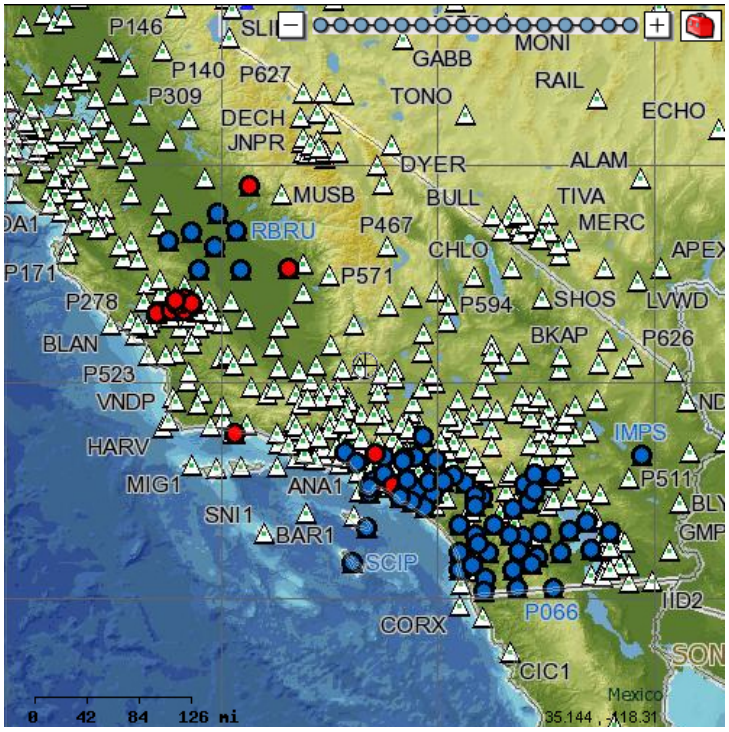

Figure 2– California Real-Time GPS Network (CRTN). Note the Continuous GPS Stations (CGPS) are depicted as triangles while the Real-Time stations are represented as circles. Image is obtained from SOPAC GPS Explorer at http://sopac.ucsd.edu/projects/realtime/

We first review the Real-Time GPS network that acts as the initial data source in our system. This is represented in the lower left hand side of Figure 1.

Real-Time GPS Networks: Continuous Global Positioning System (CGPS) has been used in geodesy to identify long-term tectonic deformation and static displacements such as inter-seismic, co-seismic and post-seismic deformation. [18]. GPS stations can run for long periods of times without need for frequent maintenance and can communicate with the data collection points using various connection types such as Wi-Fi, modems and phone lines or fiber-optic lines. Over the last 15 years networks of individual GPS stations (monuments) have been deployed along the active fault lines, and data from these are continuously being collected by several organizations. One of the first organizations to use GPS for crustal deformation monitoring was the Southern California Integrated GPS Network (SCIGN) [17]. As first conceived SCIGN did not operate in real-time. Data were collected at a 15-30 s sampling rate and retrieved and analyzed every 24 hours to compute slowly time-varying station positions. The Scripps Orbit and Permanent Array Center (SOPAC), a SCIGN collaborator, is in the process of upgrading SCIGN and PBO CGPS stations to real-time operations as part of the California Real Time Network (CRTN) (Figure 2), as a collaboration of Orange and San Diego Counties and the southern California Metropolitan Water District. Data are collected once per second and stations positions are computed on-the-fly with a latency of less than a second. Real-time data are used for seismic and structural monitoring, hazards mitigation, and by land surveyors who require cm-level real-time positioning accuracies. For computational efficiency SOPAC divides CRTN into

real-time subnetworks of about 10 stations. Each subnetwork provides real-time position data (less than 1 sec latency) and operates at high rate (1 – 2 Hz). The raw measurements from the GPS sensors are continuously collected and locally stored by the Geodetics RTD/CLP software suite consisting of the RTD ("Real Time Dynamics" server and the CLP (CommLinkProxy) and later made available to the public via FTP sites. The RTD server also broadcasts real-time positions in an open binary format called RYO. Each RYO message contains the positions of the stations that reported for that epoch.

The data collected from the GPS stations are served in various formats as following:

- **RAW and RINEX**: For archiving and record purposes; not available in real-time.
- **RTCM**: Published real-time and no records are kept. This is useful for RTCM capable GPS field receivers.
- **Positions**: Positions of the stations. Updated and presented every second. GPS Time Series can be produced using these positions, and they can be in different averaging intervals such as hourly, daily, etc.

Perhaps the most interesting of these formats for geophysicists is the position information which can be used in scientific calculations, simulation or visualization applications. The RTD server however outputs the position messages in a binary format called RYO. This introduces another level of complexity for many client applications because the messages have to be converted from binary RYO format to a data format that the client can consume. Also, to receive station positions directly from the RTD server, clients are expected to open socket connections. An obvious downside of this approach is the extensive load this might introduce to the server when multiple clients are connected.

After the RTD server receives raw data from the stations it applies internal filters and for each network generates a message. This message contains a collection of position information for every individual station from which the position data has been collected in that particular instant. In addition to the position information there are other measurements in a message such as quality of the measurement, variances etc.

SensorGrid: To process GPS sensor streams in real-time we have developed several network-enabled filter applications and Web Services to make real-time position messages available to scientific applications. These filters are connected to each other using publish/subscribe network middleware. In summary, the core of the system is to implement filter chains that perform format conversions or otherwise process the incoming data streams that originate in the real-time network described above. These filters serve as both subscribers (data sinks) and publishers (data sources). NaradaBrokering topics (which resemble directory and file name paths on a computer file system) are used to organize different network data stream sources into hierarchies as shown in Table 2. Currently the filters are being used to support 8 networks with 80 GPS Stations maintained by SOPAC.

In our architecture, filters are small applications designed to realize simple tasks such as transforming or aggregating messages. They are implemented using the Java programming

language (as is NaradaBrokering). They can also wrap more complicated data analysis applications [24] developed in other programming languages.

We have developed an abstract filter programming interface that can be extended to create new filters. A basic filter is consisted of three parts: a NaradaBrokering subscriber, a publisher and a data processing unit. The abstract filter interface provides subscriber and publisher capabilities. Typically a filter subscribes to a specified NaradaBrokering topic to receive streaming messages, process the received data and publishes the results to another topic. However outputs need not be always published. For instance a Database Filter may only receive the station positions to insert into the database for persistent archiving. Furthermore, filters can be connected in parallel or serial for realizing more complicated tasks.

The first filters we have developed are format converters that convert the original binary messages to different formats since Geographical Information System applications require different representations of geographic data. Once we receive the original binary data we immediately publish this to a NaradaBrokering topic using a *null* filter (that is, a simple pass-through filter that does not change the message. Next, another filter that converts the binary message to ASCII subscribes to this topic and publishes the output message to another topic. This can be continued to construct filter chains. For example, we have developed a Geographic Markup Language (GML) schema to describe the GPS position messages. Another filter application subscribes to ASCII message topic and publishes GML representation of the position messages to a different topic. This approach allows us to keep the original data intact and different formats of the messages accessible by multiple clients in a streaming fashion.

The GML Schema we have developed is based on RichObservation type which is an extended version of GML 3 Observation model [19]. This model supports Observation Array and Observation Collection types, which are useful in describing SOPAC Position messages since they are collections of multiple individual station positions. We follow strong naming conventions for naming the elements to make the Schema more understandable to the clients.

We used Apache XML Beans [20] for data binding purposes and created an application that reads ASCII position messages and generate GML instances using the code generated by XML Beans. SOPAC GML Schema and sample instances are available at: http://www.crisisgrid.org/schemas. The GML-OM Schema developed for GPS station messages and a sample XML output is given in the Appendix.

GPS Station Messages and Filters: As discussed above, station messages collected from GPS stations have several sub-sections. We have developed several filters that simplify or convert the messages since not all the parts of a position message are needed by most clients. Figure 3 shows the entire system including the GPS networks, proxy server, filters and the broker. The *ryo2nb* filter is used to make TCP/IP connections to the RTD server and publish these messages (which are in RYO format) into the messaging system.

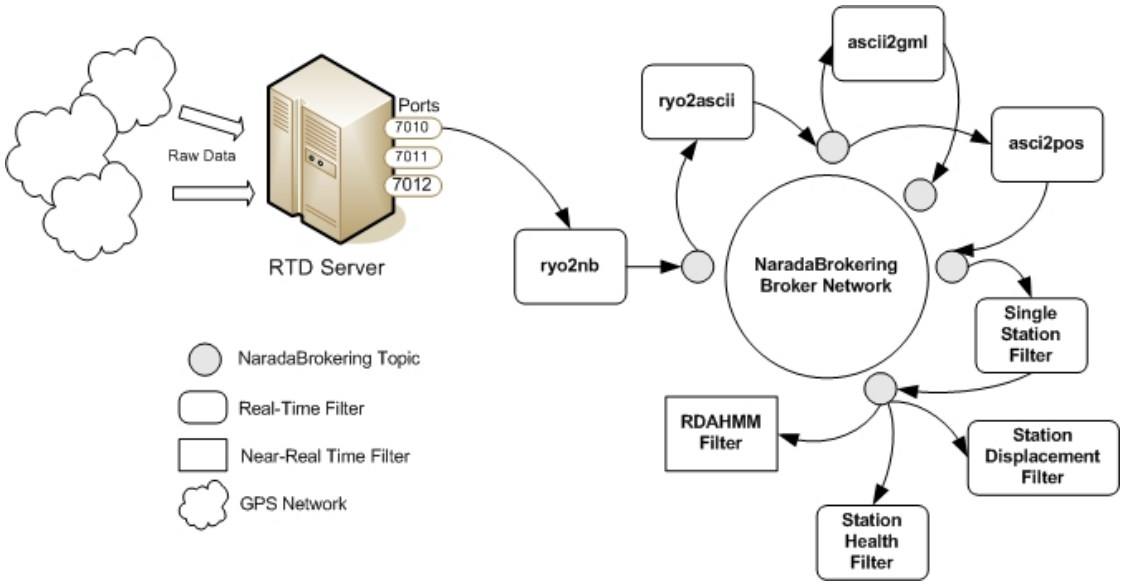

Figure 3 – Real-time filters for processing GPS streams. The various filters depicted in the figure are discussed in the text.

Ryo2ascii filter converts the whole RYO message and does not filter out anything. However some of the information included in a position message is unnecessary for most clients. For instance we have developed a user interface to display the current positions of the stations on a map. For this particular application we only need station names and their positions in terms of latitude and longitude. For this client interface we have developed ryo2pos filter (not shown in the figure) which converts RYO messages to simple position messages. Following is a sample output message from ***ryo2pos*** filter:

```
LEMA   2005-12-12 03:58:37PM-EST   36.29202028791537
-119.78237425030088     35.90217063464758
```

Here we only include (in order) the station name ("LEMA"), date, time, latitude, longitude and height values in the message. This small application is an example of how individual filters can be chained using NaradaBrokering to achieve specific tasks. Another example application integrated using this approach is RDAHMM which only requires latitude, longitude, and height values for a given station. We can easily write a filter to strip unwanted parts from the message and output only the position information.

The following table shows information about these real time networks. Note each server address is associated with a specific port number on the proxy server and publishes position data for several individual GPS stations.

Table 1 – Real-Time GPS Subnetworks, individual stations and RTD server information

Network Name	RTD Server Address	GPS Stations
CRTN_01	132.239.152.69:5010	DSME, OGHS,SIO5, RAAP, DESC, P472, POTR, P473, P483, NSSS, SCIP
CRTN_02	132.239.152.69:5011	MONP, P483, DHLG, GLRS, SLMS, USGC, CRRS, P486, P066, P480

CRTN_03	132.239.152.87:5012	PMOB, RAAP, DVLW, AZRY, DVLE, BILL, MVFD, P482, P474, P478, ESRW
CRTN_04	132.239.152.87:5013	AZRY, PSAP, KYVW, PIN1, PIN2, COTD, WIDC, IMPS
CRTN_05	132.239.152.87:5014	OEOC, SBCC, WHYT, CNPP, MLFP, MAT2, PPBF, MJPK
CRTN_06	132.239.152.87:5015	OEOC, TRAK, FVPK, BLSA, VTIS, CSDH, HBCO, CIT1, CCCS, SACY, CAT2
CRTN_07	132.239.152.87:5016	UCLP, LAPC, EWPP, LORS, CIT1, TABL, AZU1, VDCY, DYHS, GVRS

When the data is introduced into the SensorGrid architecture from the Proxy Server, we associate it with a topic. Topics are structured to identify not only the network name but also the format of the data. The following table shows the NaradaBrokering topic names for two filters:

Table 2 NaradaBrokering topics for GPS streams

Network Name	RYO Topic (null filter Publishes to)	ASCII topic (ryo2ascii filter Publishes to)
CRTN_01	/SOPAC/GPS/CRTN_01/RYO	/SOPAC/GPS/CRTN_01/ASCII
CRTN_02	/SOPAC/GPS/CRTN_02/RYO	/SOPAC/GPS/CRTN_02/ASCII
CRTN_03	/SOPAC/GPS/CRTN_03/RYO	/SOPAC/GPS/CRTN_03/ASCII
CRTN_04	/SOPAC/GPS/CRTN_04/RYO	/SOPAC/GPS/CRTN_04/ASCII
CRTN_05	/SOPAC/GPS/CRTN_05/RYO	/SOPAC/GPS/CRTN_05/ASCII
CRTN_06	/SOPAC/GPS/CRTN_06/RYO	/SOPAC/GPS/CRTN_06/ASCII
CRTN_07	/SOPAC/GPS/CRTN_07/RYO	/SOPAC/GPS/CRTN_07/ASCII

Similarly the ryo2pos filter subscribes to the appropriate RYO topic and publishes to for instance /SOPAC/GPS/CRTN_01/POS topic.

Here we give brief overview for some of the filters we have developed for SensorGrid architecture:

ryo2ascii filter: Subscribes to the RYO topic to receive binary messages, converts them to simple ASCII format and publishes to another topic (i.e. "/ASCII").

ascii2gml filter: Geography Markup Language is perhaps today's most popular geographic data format produced by Open Geospatial Consortium. We have developed a GML Schema conformant with the latest Observations and Measurements (O&M) [15] extension to describe GPS station messages. This filter converts the ASCII position messages into GML and publishes to a new topic (i.e "/GML"). We expect that in the near future most GIS applications will be developed to conform to OGC standards and presenting GPS messages in GML will help us easily integrate scientific applications.

ascii2pos filter: The RYO message type contains several sub-parts other than physical position of the station such as position quality and several optional blocks. However most of this extra information is not required by the applications. This filter eliminates optional blocks and unnecessary information from the ASCII messages to create concise position messages which only include a time stamp, station ID and position measurements.

Station Displacement Filter: One of the use cases of GPS stations is to detect seismic activities. We have developed a simple filter that analyzes position information of a GPS Station and outputs its real-time physical displacement. The filter allows displacements to be calculated based on different time intervals, i.e. actual displacement of the station in last hour or in last 24 hours.

Station Health Filter: One advantage of dealing with the real-time measurements is that we can instantly see if any of the sensors in a network is not publishing position information. We have developed this filter to log the down times of the stations and (potentially) alert administrator if a threshold value is reached. For instance it can be tolerable for a GPS station to be down for a few minutes due to network problem, but if a station has not been publishing position values for over an hour, a maintenance call may be required.

Single Station Filter: As mentioned above the original messages imported from the RTD server contain position information for multiple stations. However some applications may be required to analyze data for a particular station. For this reason we have developed this filter to pull measurements from a particular station.

RDAHMM Filter: RDAHMM (for Regularized Deterministic Annealing Hidden Markov Method) is a data analysis and classification system. This algorithm and the associated filter are extensively described in [24]. In brief summary, the RDAHMM filter can be used to classify sequences of the real-time data series and can potentially be used for mode change detection.

3. Testing the Real-Time GPS Data Grid Implementation

In the proceeding sections, we have described the implementation of the SensorGrid architecture for managing GPS data streams. The implementation is built using topic based publish-subscribe messaging system and filter Web Services. In this application domain the GPS streams are made available to the users through a series of filters connected by the NaradaBrokering messaging substrate. Our system consumes data from 8 GPS subnetworks that collective contain 80 permanent stations and make up the California Real Time Network. The GPS stations publish their positions regularly once per second.

In the simplest setup the system is comprised of a broker and several filters, so the system performance will be mainly affected by the performance of the broker since the filters will mostly be deployed on different servers. However in some cases where large numbers of filters are run on the same server performance degradation can be expected.

The messaging broker in the system is responsible for routing the real-time streams from sources to the subscribers. Since the data is continuously flowing in 1Hz frequency we want the messages to be delivered in less than 1 second before the next message is received, and we do not want any kind of queuing to delay the message delivery. Any queuing lasting more than one second or temporary storage of the messages will be detrimental since new messages will arrive continuously and the queue will continue to grow, causing delivery failures.

Therefore the performance tests should focus on finding out the maximum number of real-time providers and clients a single broker can support without introducing additional overhead or

become unresponsive. There exist limits for the broker in terms of the supportable numbers of publishers or subscribers as well as a maximum data rate.

3.1. Test Methodology

To test the performance of the system we have created a basic setup which consists of several filters and a single broker. In this setup we have three filters: a message-forwarding filter to route GPS messages from the RTD server to the NaradaBrokering server, an RYO to ASCII converter and a simple client filter. In the normal operation we connect to the SOPAC RTD server to receive the GPS messages. However, for the performance tests we recorded the raw GPS messages of a single network for 24 hours and replayed them using a store-and-replay filter. Using stored values allows us to make our tests reproducible. Also, as we will discuss, we can use these to simulate much larger networks than currently exist.

We have written two filters for recording and replaying the binary RYO messages: RYO Recorder filter and RYO Publisher filter. The first filter subscribes to a RYO topic and creates daily GPS records by saving incoming messages into files. It creates a new file after midnight, and names it to reflect of which GPS network it holds the records and for which date. For instance a file named `CRTN_01-09_11_2006-12_00_00_AM.ryo` has RYO records of the CRTN_01 network for the date 09/11/2006 and the first sample was collected at 12:00:00 AM.

In our performance tests we use RYO Publisher filter to publish the binary messages in these files to a broker topic. This way we are replacing the actual RTD server with a filter, which allows us to create as many GPS network as we want. The RYO Publisher filter also provides capability to change the message frequency. Currently the actual RTD server publishes network messages at 1Hz frequency, i.e. one message per second is published for each network. By changing a filter parameter we can change this frequency and hence the data flow rate. Considering the fact that in the near future the GPS stations are expected to work on 2Hz frequency, i.e. send their positions twice in a second, this capability of the RYO Publisher filter allows us to simulate future GPS networks.

Overall performance of the system can be estimated by measuring several characteristics:

- The stability of the system for continuous operation;
- Ability to support multiple data sources;
- Ability to support multiple clients;
- End-to-end message delivery times; and
- Ability to preserve the order of the incoming messages.

Figure 4 depicts the basic system configuration for the performance tests. The test system consists of three filters and a NaradaBrokering server. This is the simplest filter configuration that allows clients to access the GPS messages in human readable format.

The first filter is the RYO Publisher, which replaces the RTD server used in real-world operation with our archived data. The RYO Publisher filter reads a daily RYO archive file and publishes the GPS position messages to a broker topic in 1Hz frequency. The RYO to ASCII

Converter filter converts the binary messages into ASCII format and publishes to a new topic; finally the Simple Filter subscribes to this topic and receives them.

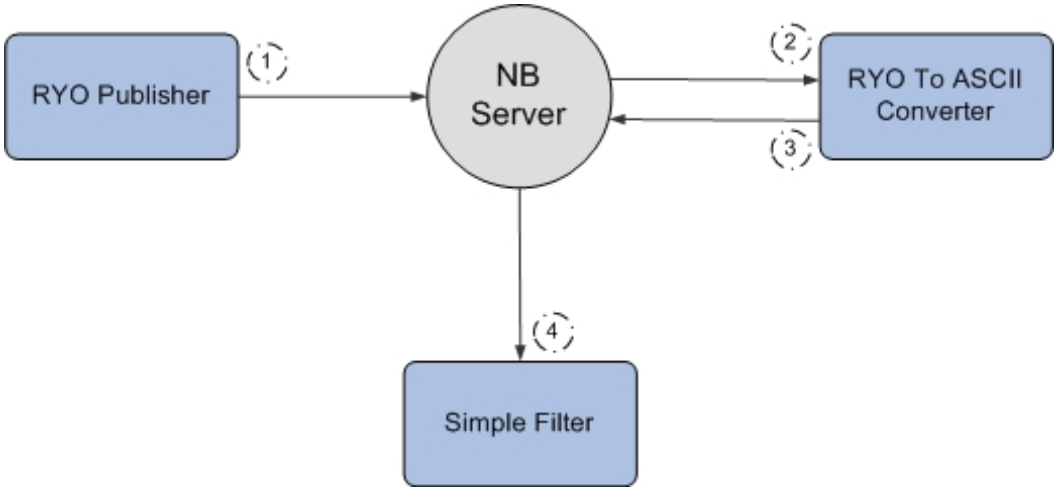

Figure 4– The basic SensorGrid performance test setup includes three real-time filters and a broker.

To measure the mean end-to-end delivery time for messages we take measurements at 4 points, as indicated in the figure:

1. Before the message is published by the RYO Publisher;
2. As soon as it is received by the RYO to ASCII converter filter;
3. After the format conversion and right before publishing to another topic; and
4. When it is received by the Simple Filter.

The configuration in Figure 4 has a complete network path from 1 to 4, but it also includes RYO to ASCII conversion between 3 and 4. So in order to find the actual wire transfer times we subtract the format conversion cost from the total time:

$$T_{transfer} = (T_2 - T_1) + (T_4 - T_3)$$

Other than the network delay, we also test to make sure that the messages are delivered in the correct order. To do this the RYO Publisher marks the outgoing messages in increasing order. It also records the message size, which may affect the transfer time.

We use built-in NaradaBrokering event properties to pass the timestamps and other information from one filter to another. To do this the RYO Publisher creates a string with three values and inserts it as *MSGSTAMP* property (a built-in property) into the NaradaBrokering Event it is about to publish. The first value is the message number, the second number is the size of the message in bytes and the last value is the time stamp in milliseconds. When the subsequent filters receive the NaradaBrokering Event, they first extract the *MSGSTAMP* property and append the current time stamp. This way all publish and subscribe operations in the filter chain will be marked in the *MSGSTAMP* property. When the final filter receives a message it extracts the string then appends its timestamp and saves it in a file for further analysis. Two message stamp samples are given here:

Message Number	Message Size (Bytes)	Time Stamp 1	Time Stamp 2	Time Stamp 3	Time Stamp 4
1	175	1159247077749	1159247078314	1159247078349	1159247078472
2	1561	1159247079030	1159247079034	1159247079058	1159247079063

Table 1 – Time Stamps Created for Performance Tests

To measure the five characteristics of the system as described above we identified following test cases:

- Stability of the system for continuous operation;
- Number of GPS networks that can be supported by a single broker;
- Number of clients that can be supported by a single broker; and
- Number of topics that can be supported by a single broker.

These tests were performed on Linux servers in the Community Grids Laboratory. Since these servers were on open networks and not dedicated to our use during our tests, we occasionally experienced delays unrelated to our tests. To eliminate these outliers in the final measurements we recursively apply a Z-filter. Given a number of measurements the Z-filter finds if any particular value is an outlier by using its standard deviation value and the average value of all the entries. For a measurement (x) the formula for the z-filter can be expressed as:

$$Z\_value = abs[t - average] / t_{standard\_deviation}$$

Then the calculated value is compared to a cutoff value, which is usually set to 2.5. That is, we discard values more than 2.5 standard deviations from the average. If the z-value is greater than the cutoff value then it is considered an outlier and removed from the measurements.

3.2. Test Results

3.2.1. System Stability Test

The first test is to verify that the system performance is time-independent and specifically does not degrade over time. Performance degradation could result from poor thread, socket, and object management in our code as well as problems with the Java Virtual Machine or operating system. To perform this test, we run the system shown in Figure 4 for 24 hours and record the timings. At the end of the test we first measure the average message delivery times, and then by dividing the timings into segments, we determine if continuous operation degrades the system performance. We also want to verify if the messages will be delivered in the incoming order.

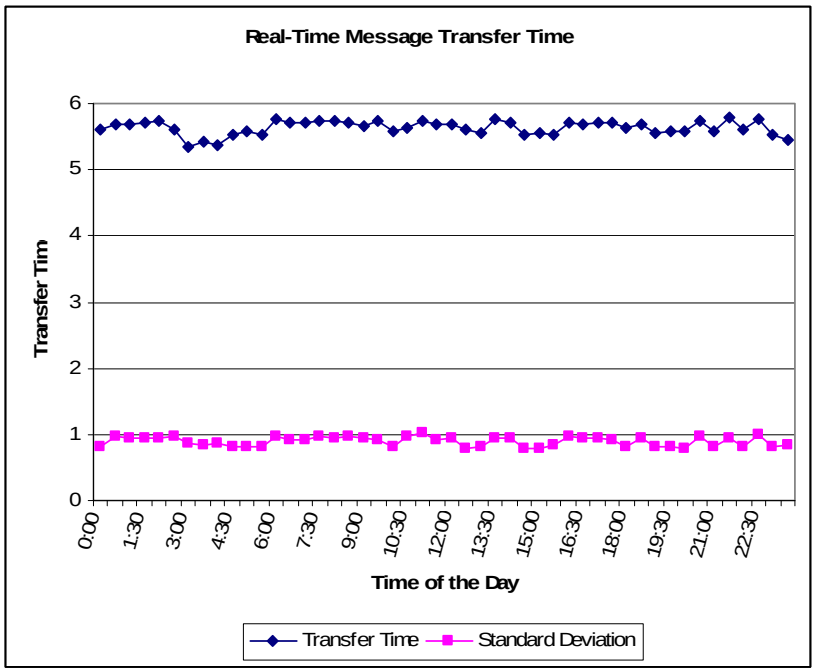

Figure 5- System Stability Test Results for 24-hour operation of the sample test setup. The transfer time (y-axis) is in milliseconds.

Figure 5 illustrates the results of the first test. The test was run for 24 hours. The last filter in the test setup described in Figure 4 records timings from all steps for each message published every second. We apply a Z-filter to clean the outlier values, and calculate averages for each half hour. Each point in the graph corresponds to 1800 measurements or roughly equal to 30 minutes worth of data. The results show that the transfer time is stable over the entire range with an average around 5.6 ms. Overall the test shows that the continuous operation results in no degradation of the system performance.

3.2.2. Maximum number of GPS networks a single broker can support

Another important feature the system should provide is to be able to serve multiple publishers simultaneously. This is important for managing GPS streams because there are multiple GPS networks we need to support simultaneously.

For this test we keep the original configuration described in Figure 4 and increase the number of the sensor data sources by adding new RYO Publishers. Thus we simulate publishing messages from multiple GPS networks. The result of this test allows us to specify the number of networks one single broker should support in the real world applications.

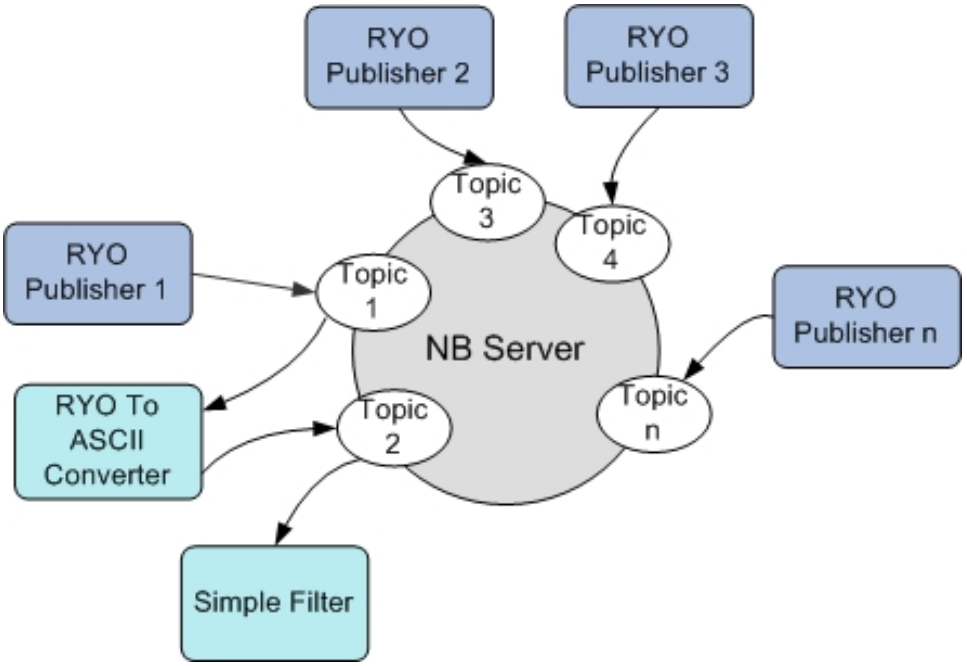

Figure 6- Multi Publisher Test Architecture is used to test the performance of *n* network publishers on a single broker.

Figure 6 shows the test architecture with additional RYO Publishers. Note the NaradaBrokering topics are used to connect filters with each other. The RYO Publisher 1 publishes the raw data to the Topic 1; the RYO to ASCII converter subscribes to this topic and receives the binary messages, which in turn publishes the converted messages to Topic 2. Subsequently the Simple Filter receives the ASCII messages from the Topic 2. For this particular test, we then start new RYO Publisher filters. Each of the new publisher filters publishes binary data to a new topic. Each publisher publishes the same (test recorded) data that was used in the previous test.

We ran this test for two consecutive days. Initially the system consisted of the components shown in Figure 4, but after every 30 minutes we started 50 new publishers. The maximum number of the publishers we were able to reach was 1000, which is due to the fact that the maximum number of open-file descriptors the operating system allowed is 1024. At the end of the test, we removed the outliers from the results and divided the results into 1800-entry segments, which is roughly equal to 30 minutes of GPS data stream.

As Figure 7 and Figure 8 show the message delivery time is always stable at around 5 ms. We do not observe any unexpected increase or decreases. This shows that even with maximum number of publishers allowed by the broker, the system supports GPS message delivery without any problems. We also confirmed that the message order is preserved during the test run. Note finally there is no system degradation over time for 1000 publishers.

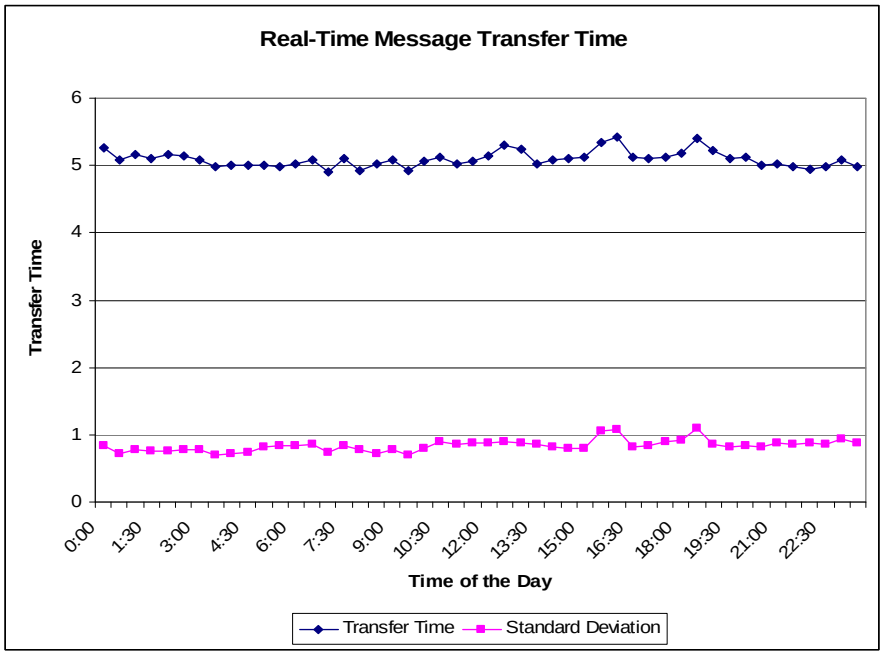

Figure 7– Multiple publisher test results for the first 24 hour period. Transfer times (y-axis) are in milliseconds.

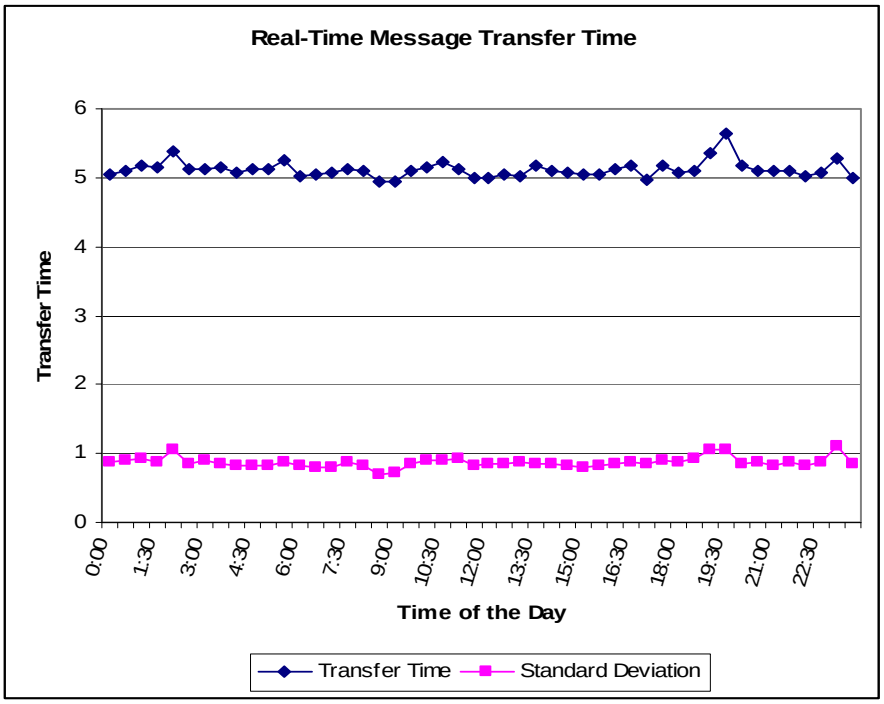

Figure 8- Multiple publisher test for the second 24 hour with 1000 active publishers

3.2.3. Maximum number of subscribers a single broker can support

In the second test described above, we explored the limits of the system from the data provider side, and found that the message broker could support up to 1000 independent GPS

station fields. In the third type of the tests described in this section, we look at the system from the client's side and try to find the number of clients the basic system described in Figure 4 can support. Determining this limit is important for providing uninterrupted real-time data access to large number of clients.

In these tests we will have only one GPS Network publishing the data and increase the number of Simple Filters to simulate the real-time data clients. The result of these tests will allow us to decide when to deploy a new broker if large number of customers plug into the real-time streams. For this test the number of clients can be increased exponentially.

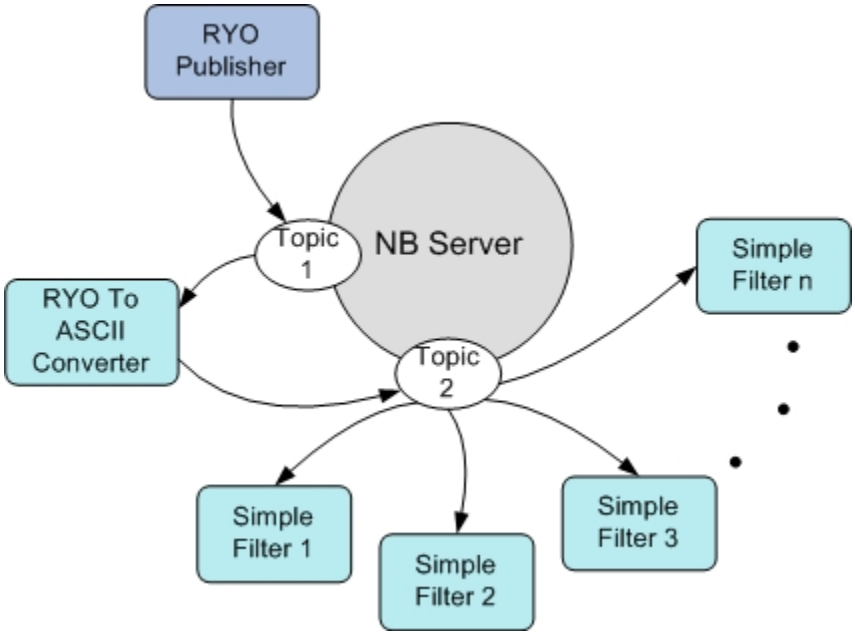

Figure 9- Multi-subscriber (client) test architecture. See text for test details.

Figure 9 shows the system architecture for Test 3. Note the RYO Publisher and the RYO-to-ASCII Converter filters are connected the same way as in Test 2. However in this particular test we run multiple Simple Filters that are all connected to the Topic 2. This allows us to see if the system can support distribution of real-time messages to large number of clients without introducing additional overhead. For our GPS applications the message frequency is 1Hz, therefore we expect messages delivered to the clients under 1000ms.

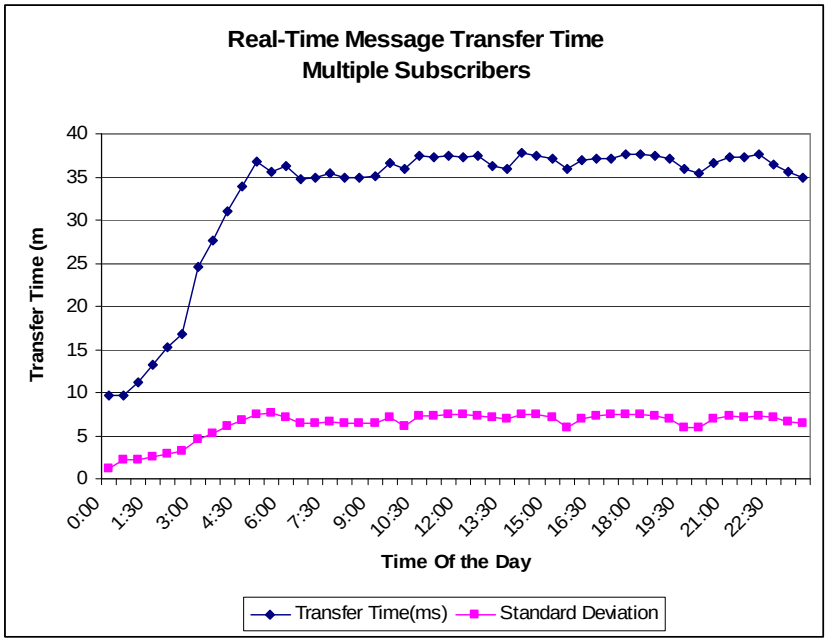

Figure 10– Multiple subscribers transfer time (in milliseconds) test results for twenty-four hours. See text for discussion.

Figure 10 shows the results of the Test 3. We begin with 100 clients. Every 30 minutes we add 100 Simple Filter clients to the system. As a result at the end of the 5th hour there are 1000 subscribers. As explained in Test 2 the underlying operating system (Linux) allows only 1024 socket connections to be open simultaneously by default, so we stop adding subscribers when we reach 1000. Therefore after the 5th hour the system works with 1000 clients for another 19 hours. We run this test for two consecutive days. The results shown in Figure 10 are the average of these two successive tests conducted for two consecutive days.

The test results tell us that the behavior of the broker for multiple clients is different than with multiple publishers. In the multiple publishers case, the average transfer time for GPS messages is almost always around 5 ms, which is same as the single publisher and single subscriber case described in Test 1. Thus we see that the number of the publishers in the system does not affect the overall performance, as long as the number of the clients do not exceed a certain threshold of the underlying operating system. On the other hand, increasing the number of clients has an obvious effect on the average message distribution time. This is due to the fact that the broker needs to forward messages to many receivers, which takes more time.

Figure 10 shows that for every 100 clients subscribing to the same topic in 30 minutes period, the average message delivery time increase a few milliseconds. The total number of clients reaches 1000 after 5 hours and the average delivery time from the publisher to the final client filter increases to 35 ms. Although this is several times higher than the average time measured in Tests 1 and 2, it is still acceptable, since the delay between successive GPS messages is 1 second. To sum up, this test shows that the system with a single broker scales up to a thousand clients with an acceptable transfer delay. Note also that performance for 1000 subscribers is stable.

3.2.4. Multiple Broker Test

The test system described in Figure 4 is the most basic configuration to provide human readable GPS messages for our application use case. The first three test cases show that this system is stable for the continuous operation and scales up to the maximum number of file descriptors allowed by the server's operating system. Both in multiple-publisher and multiple-client tests we identified the limits of the system as 1024 concurrent clients or publishers. Although this is the upper limit of the broker without changing the operating system variables for increasing open file descriptors, the system might need to support many more clients or producers. Therefore we have investigated an alternate approach for increasing the number of the clients to support. This approach is based on creating distributed NaradaBrokering broker network. NaradaBrokering allows creating broker clusters for extending message delivery. The advantage of this approach is that we can scale the system indefinitely by adding more brokers.

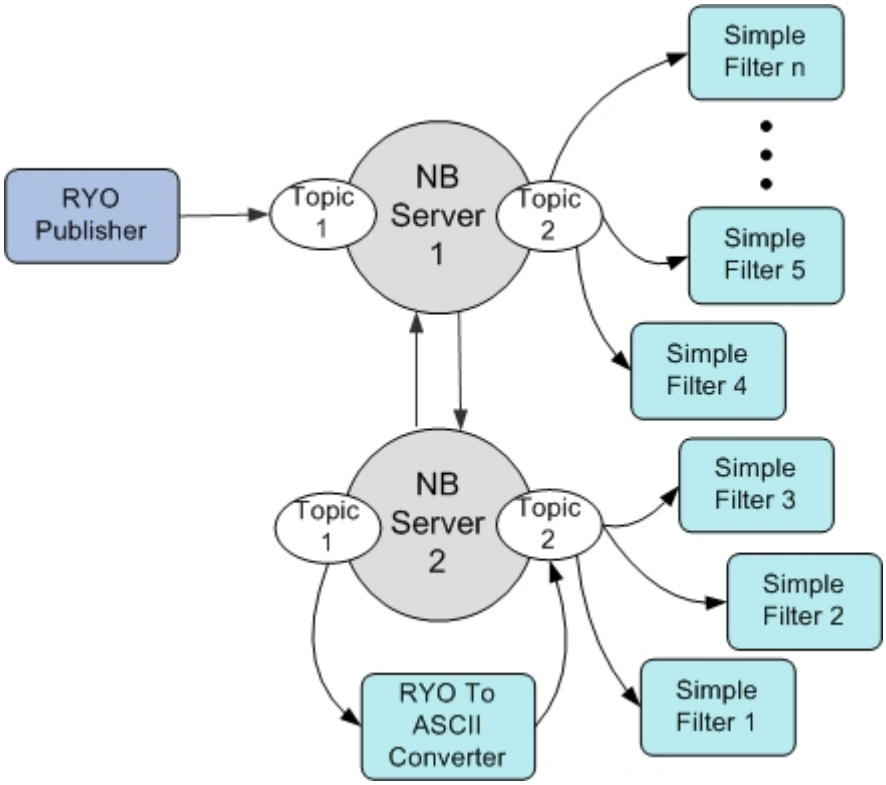

Figure 11– Multiple broker test architecture increases scalability beyond the default file descriptor limits of the underlying operating system.

Figure 11 displays the setup for multiple-broker test. For this test we first linked two NaradaBrokering servers with each other utilizing broker's networking capability. This allows brokers to exchange messages and provide access to streams on the same topics. For instance any message published to Topic-A on the first broker can be retrieved from both Broker-1 and Broker-2 from Topic-A.

This configuration potentially can solve the limits we faced in the previous tests because now we can have as many as 1024 connections on each broker. To prove that we can actually overcome the limits set by the operating system we executed this test by connecting 750 clients to each broker. Thus in total the system was run with 1500 clients for 24-hour period. We chose to run 750 clients per broker to stay well below the saturation limit of the broker. Our experiments have shown that the brokers might not work continuously with the maximum number of clients.

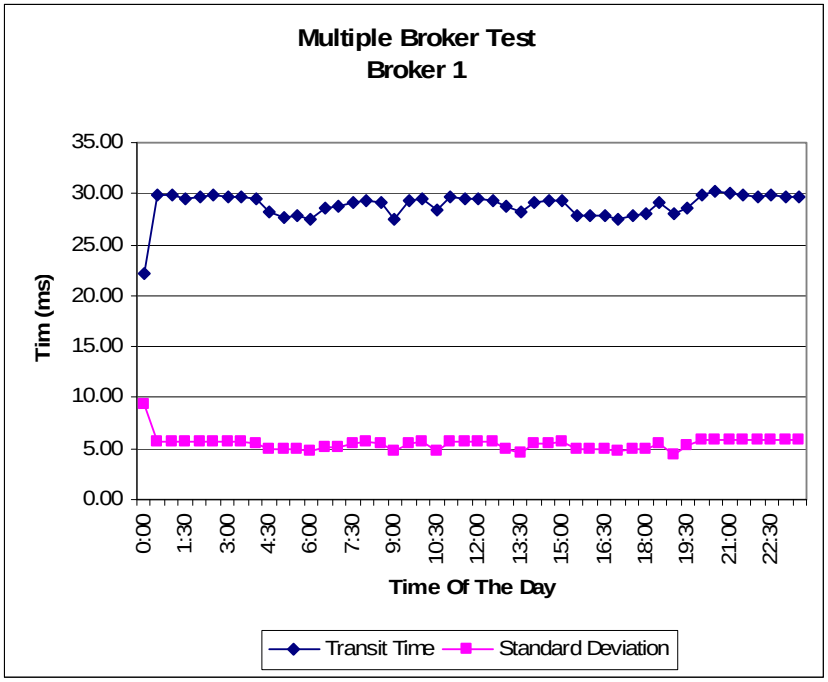

Figure 12– Total transit times for the first broker; Note that in the first 30 minutes we increase the number of the clients to 750 and the average transmit time reaches to 30ms.

To show that multiple-broker configuration is feasible and does not introduce unacceptable overheads we took timing measurements for each broker. Figure 12 shows the timings from the Broker-1, which demonstrates similar behavior with the previous single broker test. Here we see again that continuous operation does not degrade the performance.

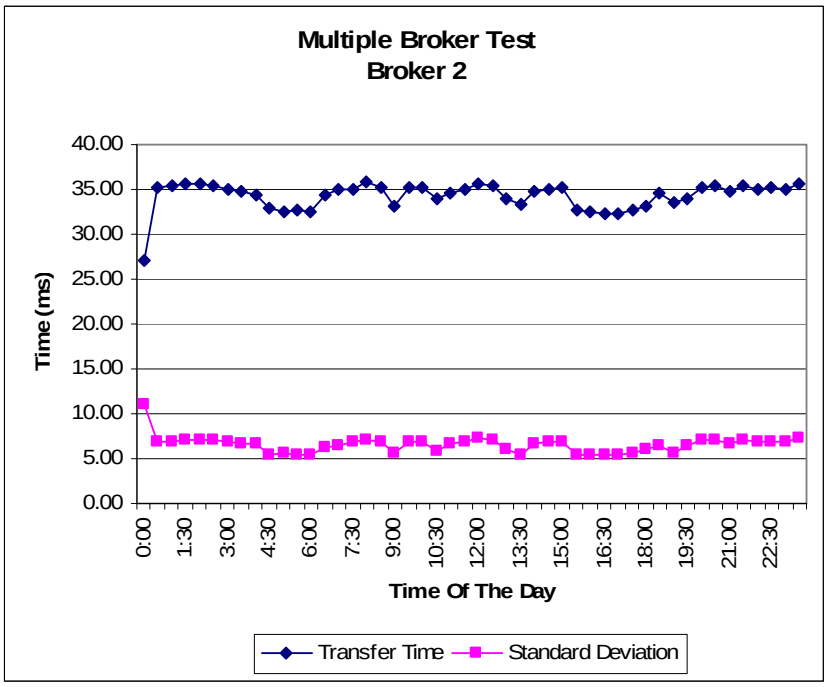

Figure 13– Total transfer times for the second broker. We start 750 clients in the first 30 minutes and the average transfer times reach to 35ms.

Figure 13 shows the test results for the second broker. Overall behavior of the Broker-2 is very similar to the Broker-1 except there is a 5 ms addition in this case. This 5 ms overhead is due to the fact that the messages published to Broker-1 is processed in this broker and sent to Broker-2. Even with this extra overhead the system performance and the 35 ms transfer time is acceptable since the continuing GPS messages arrive in 1 sec intervals. This test shows that the system can be scaled up by creating NaradaBrokering networks. This potentially means that there is no hard limit on the number of sensors or clients to support. Finally, it should also be noted that for all tests described above we have also checked and confirmed that the message ordering was preserved.

4. Summary and Acknowledgments

In this paper, we have discussed the architecture, performance, and the stability of our real-time SensorGrid architecture for GPS data. We have designed a basic filter system which represents the conventional use of our system to support real-time GPS streams. This system is currently deployed at Indiana University and utilizes data provided by SOPAC;s California Real Time Network. The basic system supports 85 GPS stations on 8 different networks and has been in continuous operation since July 2006.

We have executed several tests to find the limits of the system in terms of the number of sensors and clients that can be supported. The test results have shown that the system can be used up to either 1000 clients or 1000 sensor streams, at which point upper (default) limits of the operating system are reached. The delivery overhead introduced by our system is in all cases much less than the 1 second source data rates, and the system demonstrates no performance

degradation. To overcome the operating system limit we have designed a multiple-broker test (exploiting clustering capabilities of the NaradaBrokering messaging software) and have shown that this setup can support 1500 clients for continuous operation. The last test also shows that the system can be expanded to support as many clients or sensors as required by creating broker networks. Thus the system can support substantially larger GPS networks than are currently deployed.

This research was carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. This work was supported by the Advanced Information Systems Technology Program of NASA's Earth-Sun System Technology Office and by NASA SENH grant NAG5-13269 (Scripps) and NASA ACCESS grant NNG06GG94A (Indiana University). Use of the Geodetics, Inc. RTD software was provided by the University of California, San Diego. We thank Dr. Shrideep Pallickara for help with NaradaBrokering software.

5. References

1. Akyildiz, I.F., et al., A Survey on Sensor Networks. IEEE Communications Magazine, 2002.
2. Akyildiz, I.F., et al., Wireless sensor networks: a survey. 2002, Elsevier. p. 393-422.
3. Chong, C.Y., S.P. Kumar, and B.A. Hamilton, Sensor networks: evolution, opportunities, and challenges. Proceedings of the IEEE, 2003. 91(8): p. 1247-1256.
4. Delin, K.A., The Sensor Web: A Macro-Instrument for Coordinated Sensing. Sensors, 2002 2002. 2(1): p. 270–285.
5. Delin, K.A. and S.P. Jackson, The Sensor Web: A New Instrument Concept. 2001. p. 20-26.
6. Martinez, K., J.K. Hart, and R. Ong, Environmental sensor networks. 2004. p. 50-56.
7. Estrin, D., et al., Next century challenges: scalable coordination in sensor networks. 1999, ACM Press New York, NY, USA. p. 263-270.
8. Hudnut, K.W., et al., The Southern California Integrated GPS Network (SCIGN). 2002. p. 167–189.
9. Yamagiwa, A., Y. Bock, and J. Genrich. Real-time monitoring of crustal deformation using large GPS geodetic networks-Japanese GEONET's potential as a natural hazards mitigation system. in American Geophysical Union, Fall Meeting 2004, abstract #SF53A-0724. 2004.
10. Cardell-Oliver, R., et al., A Reactive Soil Moisture Sensor Network: Design and Field Evaluation. International Journal of Distributed Sensor Networks, 2005 2005. 1(2): p. 149-162.
11. Raicu, I. Efficient Even Distribution of Power Consumption in Wireless Sensor Networks. in ISCA 18th International Conference on Computers and Their Applications, CATA 2003, 4 pages. 2004. Honolulu, Hawaii, USA.
12. Berfield, A., P.K. Chrysanthis, and A. Labrinidis. Automated Service Integration for Crisis Management. in First Workshop on Databases In Virtual Organizations (DIVO 2004), available from http://dais.cs.uiuc.edu/divo2004/proceedings/divo04-berfield.pdf. 2004.

13. Goldammer, J.G. Early warning systems for the prediction of an appropriate response to wildfires and related environmental hazards. in Health Guidelines for Vegetation Fire Events,. 1998. Lima, Peru, .
14. Allen, R.M. and H. Kanamori, The Potential for Earthquake Early Warning in Southern California. Science 2003. 300(5620): p. 786-789.
15. Aktas, M.S., G.C. Fox, and M. Pierce, Fault tolerant high performance Information Services for dynamic collections of Grid and Web services. Future Generation Computer Systems, 2007. 23(3): p. 317-337.
16. Bock, Y., et al., Scripps Orbit and Permanent Array Center (SOPAC) and Southern California Permanent GPS Geodetic Array (PGGA). 1997, National Academy Press. p. 55–61.
17. Hudnut, K.W., et al. THE SOUTHERN CALIFORNIA INTEGRATED GPS NETWORK (SCIGN). in The 10th FIG International Symposium on Deformation Measurements. 2001. Orange, California, USA.
18. Bock, Y., L. Prawirodirdjo, and T.I. Melbourne, Detection of arbitrarily large dynamic ground motions with a dense high-rate GPS network. GEOPHYSICAL RESEARCH LETTERS, 2004. 31.
19. Cox, S. (2003) Observations and Measurements. Volume, DOI: OGC 03-022r3
20. Apache, X.M.L., Beans Project, http://xmlbeans.apache.org/. 2003.
21. Shrideep Pallickara, Geoffrey Fox: NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Middleware 2003: 41-61.
22. Shrideep Pallickara, Geoffrey Fox, Beytullah Yildiz, Sangmi Lee Pallickara, Sima Patel, Damodar Yemme: On the Costs for Reliable Messaging in Web/Grid Service Environments. e-Science 2005: 344-351.
23. Shrideep Pallickara, Marlon Pierce, Harshawardhan Gadgil, Geoffrey Fox, Yan Yan, Yi Huang. A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems. Proceedings of the 7th IEEE/ACM International Conference on Grid Computing (GRID 2006). Barcelona, Spain; September 28th-29th 2006.
24. Robert Granat, Galip Aydin, Marlon Pierce and Zhigang Qi, Analysis of Streaming GPS Measurements of Surface Displacement Through a Web Services Environment, 2007 IEEE Symposium on Computational Intelligence and Data Mining, April 1-5 2007, Honolulu, HI, USA. Preprint available from http://grids.ucs.indiana.edu/ptliupages/publications/GranatPierceAydinQi_IEEE_CIDM07_submitted.pdf.