Space Surveillance Network and Analysis Model (SSNAM) Performance Improvements

Albert Butkus
Master Solutions, LLC, Torrence, CA

Kevin Roe
Maui High Performance Computing Center (MHPCC), Kihei, HI
Kevin.Roe.ctr@mhpcc.hpc.mil

Barbara L. Mitchell
Lockheed Martin, IS&S, Colorado Springs, CO

Timothy Payne
US Air Force Space Command, Space Analysis (AFSPC), Peterson AFB, CO

Abstract

The Space Surveillance Network and Analysis Model (SSNAM) is an Air Force Space Command (AFSPC) model, which provides the capability to analyze and architect Space Surveillance Network (SSN) Force Structure. To provide these capabilities SSNAM supports two types of simulations: Catalog Maintenance, and Special Events (Launch, On-Orbit Events, and Breakup). There are many configuration options available with SSNAM: models for all the sensors currently in the SSN to include space based and ground based sensors, hours of operation by sensor, track capacity by sensor, models for sensors yet to be created, user defined weather conditions, National Aeronautical and Space Administration catalog growth model including space debris, and solar flux just to name a few.

SSNAM is a large software system. It is written in Java, C/C++, and FORTRAN (77 & 95), represents over a million lines of code, and employs a web-based, load-sharing architecture to decrease simulation runtime. Catalog Maintenance simulations are both computationally and input/output (I/O) intensive. A typical Catalog Maintenance simulation (10K to 35K satellites simulated over a 90 day period) will generate over a terabyte of data, during the course of a simulation, which is reduced down to approximately 1.5 gigabytes. Depending on simulation configuration, runtimes can range from 12 to 48 hours on a 16 node, PC network cluster.

Because of the high computational demands of SSNAM Catalog Maintenance simulations and the anticipation of transitioning SSNAM to model the maintenance of an special perturbation (SP) catalog, the SSNAM system was ported to run on Maui High Performance Computing Center (MHPCC) platforms. This port resulted in at least a three-fold increase in performance for all currently parallelized processing in SSNAM. This paper provides an overview of the SSNAM application, its web based, load sharing architecture, the effort involved with porting Java and FORTRAN to MHPCC platforms, the approach and implementation for parallelizing the SP Tasker, and the resulting performance gains.

1. Background

SSNAM is a networked computer simulation model developed under the sponsorship of AFSPC. The purpose of SSNAM is to provide an analysis model to perform "end-to-end" simulations, re-enactments, and studies of space surveillance events and missions to aid in understanding the performance, response, and processing characteristics of the SSN. SSNAM provides a capability to evaluate changes to the SSN relative to upgrades to sensors, down time of sensors, deletion of sensors, or addition of new sensors. SSNAM also provides the capability to assess the impact of catalog growth. Impact is evaluated relative to Catalog Maintenance and Special Event (Launch, Breakup, and On-Orbit) processing missions. The performance of the current system is measured via a set of recognized parameters routinely taken from daily operations.

SSNAM is specifically designed to answer the following kinds of questions:

a. What If I Shut Down a Sensor?
b. What If I Add a New Sensor or Modify an Existing Sensor?

HPCMP USERS GROUP CONFERENCE 2007 (HPCMP-UGC 2007)
0-7695-3088-5/07 $25.00 © 2007 IEEE

c. What If I Add a Space-Based Constellation Or Change Constellation Configuration?
d. What If I Move a Sensor?
e. What If I Grow the Satellite Catalog to a Future Configuration? What If I Add a Debris Catalog?
f. What If I Change Sensor Operating Hours?
g. What If I Change Tasking?
h. What If I Change Sensor Responses?

Thus, the object of the catalog maintenance simulations is to assess the quality of the satellite catalog resulting from various proposed changes to the SSN and/or catalog population. Prior to the creation of SSNAM, AFSPC manually assessed changes as a result of SSN impacts simply by subtracting benefit from contribution.

2. Motivation

In order to achieve the required model fidelity for catalog maintenance simulation, SSNAM executes the entire catalog maintenance loop as it is run in Cheyenne Mountain - only daily SSN observation input is simulated. Actual operational software catalog maintenance routines are integrated into SSNAM. These include: the daily Tasker and the Astro Standard algorithms required for catalog maintenance. Astrodynamic Standard algorithms are also used for generation of simulated daily observations.

The Figure 1 depicts the high level control flow through a SSNAM Catalog Maintenance Simulation. Each SSNAM simulation is coordinated and controlled via the SSNAM Central Server Executive (EXEC). On receipt of the Start Simulation request the EXEC first acquires and verifies the Starting Conditions for the simulation. Then the EXEC allocates, activates, and populates the computational resources designated for the simulation.

Once the computational resources are ready the Tasker is initialized and invoked to generate the tasking request for the first simulation day. This tasking request is then used by the Loop to simulate the Response to Tasking. This is done by propagating each satellite through the geometric coverage of each tasked sensor (Perfect Observation Generation), simulating B3 Observations from each pass through each sensor's coverage (Observation Thinning and Noising), maintaining an Observation database for each satellite and updating the orbital elements (Sequential Differential Corrections), and then updating the Truth model (SP state vectors) and storing critical information for Simulation Evaluation (VMAG calculations, metrics, and stats). The observations generated for each satellite are then returned to the Tasker for evaluation and for generating the Tasking Request for the next simulation day. This process is then repeated for the number of simulation days requested by the user, typically 60 to 90 days. After the simulation is completed the results from each simulation day are evaluated and then stored for later analysis.

This is a complicated problem. As already mentioned, SSNAM Catalog Maintenance simulations are computationally and I/O intensive. The goal is to reduce catalog maintenance simulation times so that 90 day simulations can be started at the end of a business day and complete over night for analysis the next business day. Porting SSNAM to the MHPCC supercomputing environment aids in this goal and will be required to model SP catalog maintenance.

The SSNAM architecture provided further motivation for SSNAM as a candidate to port to the MHPCC. SSNAM is architected using a simple web-based architecture framework—a web-based, open system design yields a programming language and platform independent system consisting of highly portable software that could be readily migrated to the MHPCC supercomputing environment.

3. Development

The initial SSNAM prototype was developed in the late 1990s and the very first execution of this model required 36 hours to simulate two days on a single, mid 90s vintage SGI workstation. With a performance ratio of 18 hours per simulation day something had to change in order to evaluate SSN changes using SSNAM in a timely fashion. Since this time, the computational and I/O demands of the SSNAM Model have increased, while at the same time, processor performance and throughput have increased. At the writing of this paper, if one SSNAM simulation day was executed on a single Hoku CPU (3 GHZ, 64-bit OPERTON) it would take on average approximately 100 minutes to complete with a performance ratio of 1.67 hours per simulation day. Although today's technology yields a significantly higher performance ratio over the original SGI workstation, it would still require over six days for a 90 day simulation to complete.

The architectural approach used within SSNAM to decrease runtime is Load Sharing. The SSNAM Central Executive divides the satellite catalog across a collection of computational nodes dedicated for SSNAM simulation runs. The Executive also level loads the distribution across the cluster by taking into account the performance characteristics of each node. Although this approach involves a relatively straight forward implementation strategy, there are a number of performance considerations which must be addressed. First, it is important to use an efficient, scalable messaging and control mechanism. Without this, the performance cost of

managing a multi-node simulation can exceed the savings offered by the node cluster. Secondly, it is important to design the software to be as platform independent as possible. From a performance perspective there are a number of reasons for preferring platform independence: being able to assemble a cluster from whatever computational resources available (usually this is a heterogeneous collection), and to be able to upgrade cluster components individually as new/faster platforms become available. The following list highlights several key design and development decisions made to achieve efficient messaging and platform independence:

- The Central and Distributed Executives communicate via a Servlet based, multi-threaded, HTTP(S)/HTML messaging application programming interface (API)
- Java is used for the framework and data management components
- FORTRAN is used for all the computationally intensive components
- Any operating system specific aspects are handled as either runtime settable parameters or via command procedures

These design and implementation choices have proved invaluable over the years. The first Load Shared implementation of SSNAM ran on one SGI (from the original prototype), one Sun, and six PCs. On later funding cycles sufficient resources were available to purchase additional PCs and now there are two SSNAM labs in Colorado Springs, each with 16 PC network clusters. Early on, we phased out the SGI and the Sun platforms because they contributed very little to decreasing runtime when compared to the much faster PCs.

4. Porting SSNAM to the MHPCC

While most of the work was accomplished in Colorado Springs, the SSNAM team made two trips to Maui to port SSNAM code to MHPCC platforms. The first trip focused on porting the Distributed Server components, with an emphasis on performance characterization. The second trip focused on fully installing SSNAM for usage from Colorado Springs and the associated security setup for the HTTPS messaging.

On the first trip, June 2005, members of the SSNAM technical team traveled to Maui in order to conduct the initial port of SSNAM to the MHPCC. The SSNAM team and MHPCC personnel collaborated on porting SSNAM to two platforms: IBM P3 and P4. The porting effort was accomplished during the first two days with the remaining time focused on optimizing and tuning SSNAM to run on other MHPCC hardware. For the initial effort only the Distributed Server components of SSNAM were ported (indicated by the light yellow shapes in Figure 2). The SSNAM Central server and SGP4 Tasker ran on a laptop brought from Colorado Springs. The diagram in Figure 2 depicts the high level SSNAM architecture and the components ported to the MHPCC hardware.

The porting effort was divided in two concurrent paths and was completed in about 1.5 calendar days. The first path focused on getting the Java Distributed Server running on the target hardware. Since this code is written in Java, and Java SE Development Kit (JDK) 1.4 was available on the target platforms, the application ran with little difficulty. An installation script was created to support various re-configurations of SSNAM in order to explore runtime optimizations.

The second path focused on getting the FORTRAN applications built, validated, and optimized on MHPCC's IBM Power4 platforms. Before the port to the MHPCC, SSNAM simulations were conducted entirely on Windows platforms. As such, the main problems encountered during the two-day porting effort were all related to the UNIX-based operating system differences with Windows, mostly case sensitivity and file separator characters. These problems manifested themselves in three areas:

- Inconsistent data file name case
- Inconsistent FORTRAN include statement case
- Java "public static final" qualifier for the file separator character

Because the primary focus was porting SSNAM Distributed Server components to MHPCC hardware, simulation runs were limited to single day tests. This maximized the amount of time available for exploring optimization approaches for full SSNAM simulation runs. The following chart summarizes the results of the single day runs.

The Figure 3 clearly indicates that using a judicious[1] number of Nodes (and CPUs per node) decreases the amount of time required to conduct a SSNAM run, that is, Load Sharing works for this type of application. One of the limiting factors encountered while exploring various approaches was the PC laptop used for hosting the SSNAM Central Server. Because the laptop was not a Windows server it was limited on the number of concurrent network connections; hence, we never ran more than 12 Distributed Server applications at one time.

[1] Judicious. Since SSNAM is both CPU & I/O intensive it is beneficial to isolate SSNAM processing on a CPU/Disk pair. However, when one node has multiple CPUs all sharing one disk a tradeoff decision emerges regarding the actual number of SSNAM Distributed Servers to install per node. Using both CPUs per Hoku node increases runtime performance by less than a 10% over using only one CPU per node. But remember that Hoku is shared by multiple users, so generally speaking, 25 nodes can be allocated for use faster than 50 nodes.

The second trip, Oct 2005, provided a full installation of SSNAM on the MHPCC supercomputer, Hoku. This installation required the addition of two Windows PC servers to host two SSNAM Central Server applications. The second server acts as a backup to the primary server and allows for running concurrent SSNAM runs. And to support the two Windows servers two Hoku nodes were reconfigured as dedicated SSNAM Central Server proxies, this done in support of MHPCC security requirements. Finally, in order to meet the no-clear-text-messages security requirement all SSNAM web protocols were augmented to use HTTPS (and Java Secure Socket Layer APIs) in order to encrypt the inter-server, clear text HTML messages. It is now a runtime switch in SSNAM whether of not to use secure sockets for communication.

Figure 4 depicts the Primary processing components in SSNAM and their respective contribution to the overall time for a given simulation: the Tasker (in this case the GP Tasker), the Loop, and Evaluation. The timing metrics for this chart are from a baseline SSNAM simulation designed specifically to compare the performance difference between Hoku and the PC Cluster in Colorado Springs. This 60 day simulation consisted of the GP Tasker[2], the Space Surveillance Network as it existed in 2005, and a 10k satellite catalog.

The Hoku 50 run only required allocating 25 nodes. The SSNAM Central Server installs one SSNAM Distributed Server on each of the two CPUs available per node. Figure 4 shows that the Hoku 50 ran the Loop processing more than three times faster than the COS 16 PCs. This reveals good linear scaling with the Load Sharing framework. However, the Figure 5 reveals that 50 CPUs, for this particular SSNAM simulation, is the maximum number of CPUs which can be allocated before linear scaling begins to break down.

The reason for the breakdown in linear scaling is due to the technique used to manage the inter-server messaging: each server is commanded one at a time from a single thread. A prototype was conducted on this year's funding in which the inter-server messages were managed in separate threads thereby lowering the overhead substantially. Initial results look promising in maintaining linear scaling if all the CPUs, on all the nodes on Hoku are used (over 250 CPUs). This modification is planned for next year's funding.

5. Parallelizing the SP Tasker

For SSNAM to continue to model accurately the current operational environment, the SP Tasker was integrated into SSNAM on the 2006 funding cycle. In the operational environment the SP Tasker is hosted on an SGI platform and requires between one to three hours of processing time to execute depending on various configuration options. When the SP Tasker executes on the SSNAM Windows server it takes approximately 30 minutes when run against a 10K satellite catalog. At this rate it requires over 45 hours of processing time just for the SP Tasker. Parallelizing the SP Tasker became necessary to keep catalog simulation times reasonable.

The SP Tasker software architecture is structured to readily facilitate parallelization of one its primary functional areas: Probability of Detection (PoD), which takes well over 80% of the overall processing time. Since the PoD calculations are generated for a predetermined set of sensors these calculations are independent can therefore be parallelized using the same load sharing technique as other parts of SSNAM. The Figure 6 chart depicts the performance gains of load sharing the SP Tasker.

6. Summary

Porting SSNAM to run on MHPCC computational resources has proved beneficial for the SSNAM user community. This effort allows SSNAM simulations to be completed in half[3] the time required to run the same simulation on one of the Colorado Springs PC clusters. However, with the dual Central Server configuration two SSNAM simulations can be run concurrently. Running concurrent SSNAM simulations on Hoku has the net effect of decreasing run times by 75%.

[2] The SP Tasker is being integrated in SSNAM under FY06 funding and as part of the integration effort it is being restructured to use the SSNAM Load Sharing framework. We anticipate presenting the results of this effort at AMOS 2007. At the time of writing this paper the only SSNAM component which is Load Shared is the Loop processing.

[3] Remember, as of the writing of this paper the only load shared SSNAM component is the Loop processing; the GP Tasker is single threaded and, as figure 4 indicates, requires about the same time in Colorado Springs as it does on Hoku. The 16 CPU Loop in Colorado Springs runs about 3.5 times slower than the 50 CPU Loop on Hoku.

0-7695-3088-5/07 $25.00 © 2007 IEEE

SSNAM High Level Control Flow

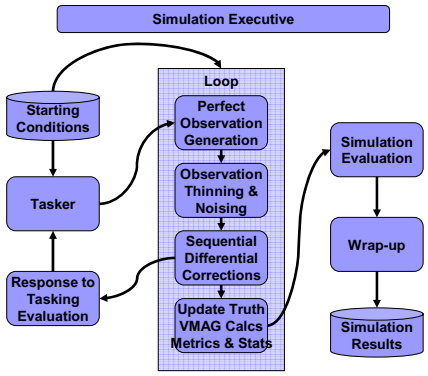


Figure 1

SSNAM Web Based, Load Sharing Architecture

- Key Standards
  - HTTP, HTTPS, HTML, and Servlets
  - Astro Standards & SGP4 Tasker
  - Standard data formats (ASCII, TLE, B3, etc)
- Platform Independent
  - Java & FORTRAN 77
- Scalable
  - Load sharing
- Modular
  - Encapsulation of data and FORTRAN applications using Early Specification for Servlets
- Yellow components ported to Hoku, blue components run on Windows Server

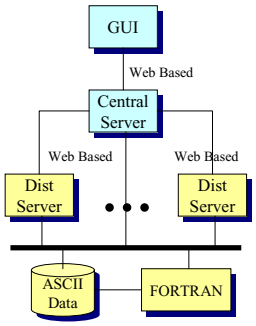


Figure 2

Four Initial SSNAM Runs on MHPCC Platforms

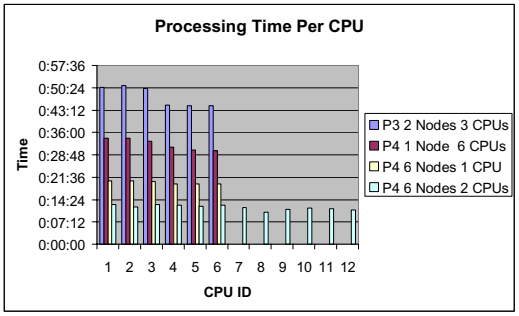
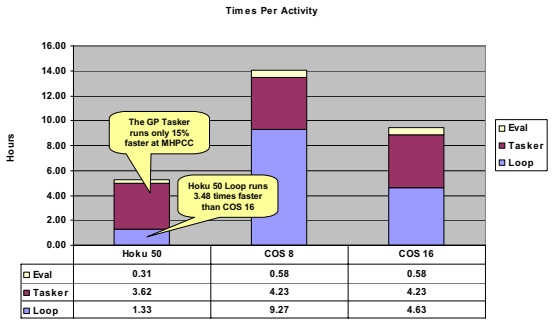


Figure 3

Major Processing Components

SSNAM code has 3 major sections: Tasker, Loop, and Evaluation



	Hoku 50	COS 8	COS 16
Eval	0.31	0.58	0.58
Tasker	3.62	4.23	4.23
Loop	1.33	9.27	4.63

Figure 4

Cost to Administer CPUs

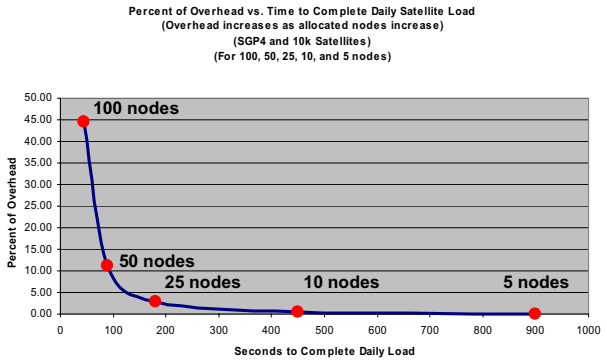
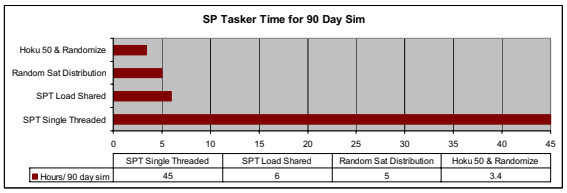


Figure 5

SP Tasker Performance

- In the operational environment the SP Tasker requires approx. 3 hours per actual day
- In the SSNAM simulation environment (10k Satellite, 90 day simulation):
  - **In the COS** lab the SP Tasker requires 30 minutes per sim day or 45 hours for a 90 day sim
  - **In the COS** lab the SP Tasker (load shared) requires 4 minutes per sim day or 6 hours for a 90 day sim, or 5 hours when satellites are randomized
  - **On Hoku** (50 CPUs) we anticipate about 2:40 minutes per sim day or approximately 4 hours for a 90 sim, or 3.4 hours when satellites are randomized



	SPT Single Threaded	SPT Load Shared	Random Sat Distribution	Hoku 50 & Randomize
Hours/ 90 day sim	45	6	5	3.4

Figure 6

0-7695-3088-5/07 $25.00