SCALABLE TRAJECTORY DESIGN WITH COTS SOFTWARE

Kenneth Kawahara[(1)] **and Jonathan Lowe**[(2)]

[(1)]*Analytical Graphics, Inc., 6404 Ivy Lane, Suite 810, Greenbelt, MD 20770, (240) 764-1500 x8534, kkawahara@agi.com*
[(2)]*Analytical Graphics, Inc., 6404 Ivy Lane, Suite 810, Greenbelt, MD 20770, (240) 764-1500 x8505, jlowe@agi.com*

Abstract: *Leveraging scalable computing architectures for trajectory design applications can provide substantial benefits to the traditional mission design process. Commercial Off-The-Shelf (COTS) software making use of this technology is now available from Analytical Graphics, Inc. (AGI) through the Systems Tool Kit (STK) Server. STK Server's two main functions are to give mission designers the ability to create, manage, and distribute mission models and analysis services over the Web or a network, and to provide a framework for distributed or cluster computing. We apply STK Server's distributed computing capabilities towards orbit design problems using STK Astrogator to quantify performance and productivity improvements. Sample trajectories for geocentric and lunar missions are used in these evaluations.*

First, we enhance the performance and usability of large-scale, computationally intensive, or repetitive analysis tasks for the mission designer through distributed computing. A reference implementation for Monte Carlo analysis uses an orbit covariance matrix to create state perturbations for thousands of propagations. Second, we implement a numerical search algorithm used in Astrogator Targeting that distributes parameter evaluations for improved performance.

Keywords: *cloud computing, trajectory design, STK, Astrogator, Server, scalable*

1. Introduction

Leveraging scalable computing architectures for trajectory design applications can provide substantial benefits to the traditional mission design process. We identified two common analyses to investigate in a COTS scalable, distributed computing environment.

Monte Carlo simulations are a standard type of analysis performed by trajectory designers to characterize the effects of probabilistic uncertainties in independent variables. These simulations are good candidates for benefitting from parallelization and scalable computing since they typically consist of thousands of independent function evaluations.

A more frequently used analysis which may also benefit from parallelization is a numerical search or optimization algorithm used to optimize aspects of a trajectory, such as the timing, direction, or size of a maneuver. Although varying greatly with the algorithm type, these typically have fewer function evaluations than Monte Carlos and usually have some dependency between the various iterations.

1.1 STK Server

STK Server provides a method of distributing computationally intensive or repetitive tasks. There are several main elements that support this process: tasks, jobs, the coordinator, agents, and agent hosts. Tasks are the building blocks of STK Server. In the simplest terms, a task is a function that gets executed by STK Server. Jobs consist of one or more tasks and allow groups of tasks to be managed as a unit. Job requests are sent to the coordinator, whose role is to manage the overall flow and execution of the job by assigning the job's tasks to the agents and hosts that are connected to the STK Server.

Agents and hosts are the elements that execute the function of a given task. In general, agents are individual machines (desktop, laptop, server) while hosts are associated with the number of cores that are available on that specific machine. For example a quad-core machine has from one to four hosts. While it is possible to set the host count to a number greater than the number of physical cores on an agent, performance gains may be minimal. Figure 1 is an illustration of the overall architecture of STK Server.

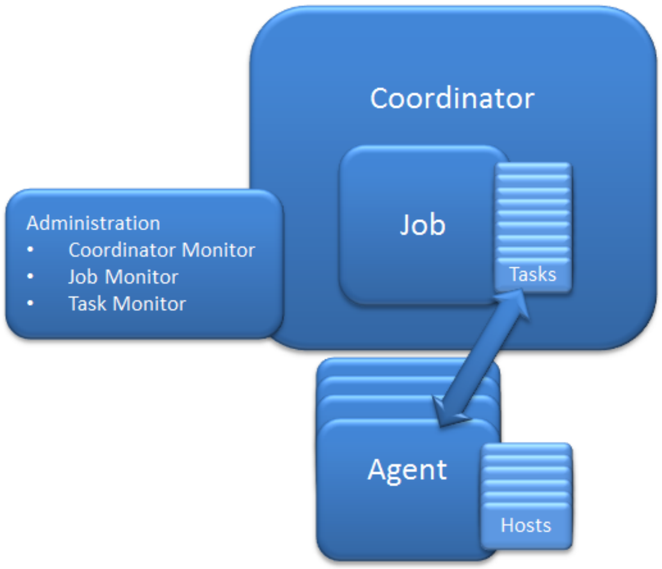

Figure 1. STK Server Architecture

Jobs are assigned a task environment. Task environments consist of a unique ID and functions that define how a host needs to be setup in order to properly run a task. For example when a newly connected agent receives tasks from the coordinator, it will instruct its hosts to configure themselves based on the job's task environment. The task environment may also define whether the task should run using the 32-bit or 64-bit STK Engine. Each host retains this state, ready to receive and quickly execute additional tasks without completely re-initializing. If the agent receives a new type of task, it will instruct the agents to recycle. Hosts will also recycle if they reach a user-defined task per host count.

A suite of administrative tools are included with the desktop implementation of STK Server to allow for setup, monitoring, and debugging. The coordinator monitor provides information on the status of agents that are available and currently connected to the Server. From the coordinator monitor, the designer can enable/disable the use of a particular agent, set agent priority, and view the number of parallel processes available on that agent. The job monitor provides summary information on jobs that have been submitted including the progress, runtime, submitter information, and priority. Finally, the task monitor provides summary information on all of the individual tasks that have been submitted including the progress, run time, and the agent assigned to the task.

1.2. STK Astrogator and UI Plugins

STK Astrogator allows users to model complex mission trajectories by assembling a series of segments within a Mission Control Sequence (MCS)[1]. The target sequence segment allows users to automate certain aspects of define trajectory design problems through profiles, which act on or change certain segments in the MCS. One type of profile is an iterative search algorithm, which exposes independent control variables, such as the ΔV magnitude of an impulsive maneuver, and segment results, which act as the dependent variables in the design problem. In addition to the three installed algorithms (Newton-Raphson, Broyden's Method, and Design Explorer Optimizer), an API is provided for a generic user-supplied search profile algorithm, which will run in-process with and is driven by the MCS execution. These Search Profile Plugins are one example of a number of available Engine plug-in points, which allow the addition of user-supplied code which will run in-process with STK.

STK's User Interface (UI) plugins give users the ability to incorporate custom code and user interfaces seamlessly into STK's graphical user interface (GUI) in the form of toolbars, context sensitive pull down and right-click menus, and GUI panels. The UI plugins have full access to STK's application programming interfaces (API), Connect and Object Model, which allow for automation and programmatic access to STK properties and functions. Figure 2 illustrates the overall architecture of the STK application, including the UI and Engine plugins.

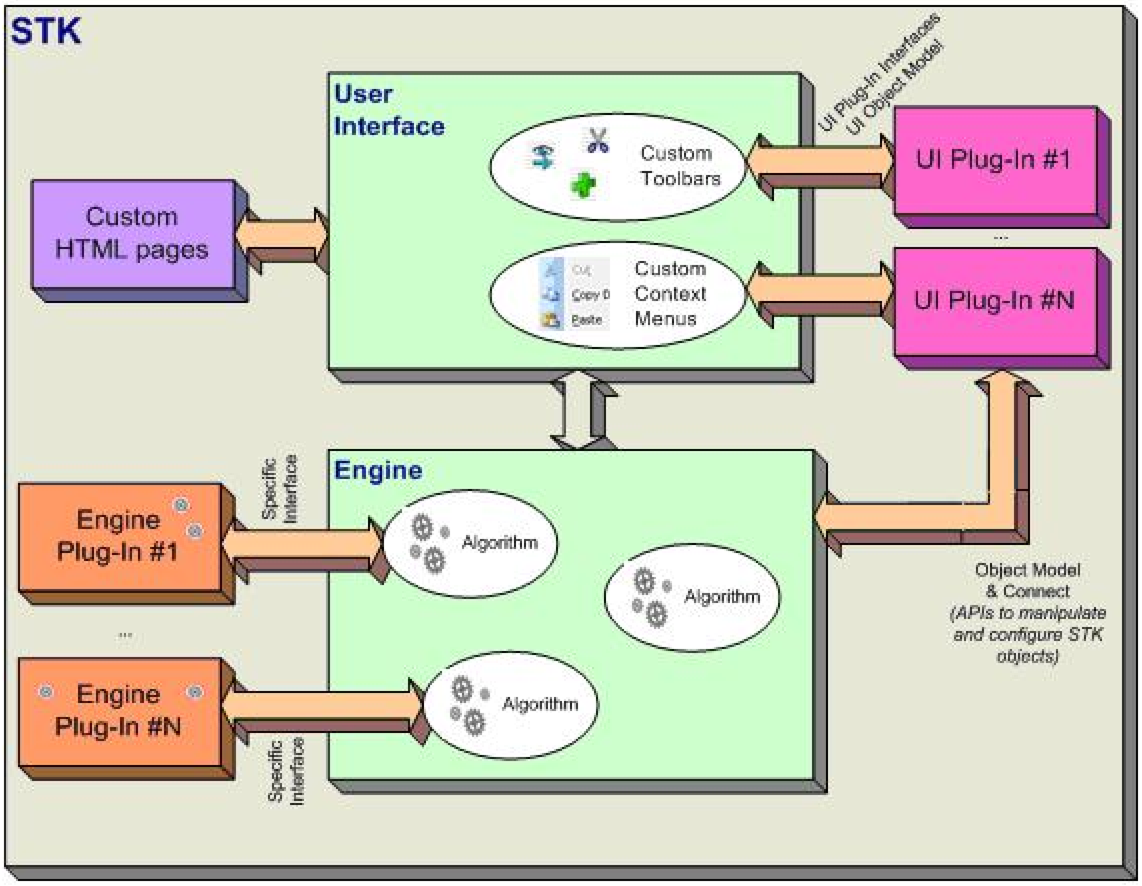

Figure 2. STK Application and Plugin Architecture

2. Monte Carlo Implementation

In this example, we explored the effect of distributing the iterations of a Monte Carlo simulation. The example scenario was a representative Lunar mission that originated in a LEO parking orbit, performed an impulsive trans-lunar insertion maneuver, and propagated until periselene. The UI plugin allows the user to create perturbations to the spacecraft's initial state by using a 6x1 random Gaussian number generator in conjunction with a user-defined covariance matrix. The UI plugin also gives access to the STK Server architecture. Upon start of the simulation, the UI plugin saves the scenario to a network location available to all of the agents. All of the tasks are formed using the perturbed states generated above and are submitted to the coordinator within a job.

The job's task environment instructs hosts to start STK and load the saved scenario which has the unperturbed initial state. Each task then updates its local copy of the scenario using the perturbed state and propagates the satellite until periselene using the original ΔV from the unperturbed case.

Test cases using 1000, 5000, and 1000 iterations were executed using two agent machines running STK 10.1 on Windows 7. Table 1 summarizes their system specifications. Each test case was first run using 1, 2, or 4 cores on each machine independently in order to compare their relative run times. The test cases were then run using all cores on both machines in parallel.

Table 1. Summary of Agent Specifications for Monte Carlo Application

	Physical Cores	CPU Speed (GHz)	RAM (GB)
Laptop	4	2.5	8
Desktop	4	2.7	6

Figures 3 shows the run times of each test case based on their configurations and the relative performance increase of adding additional processors. Note that in all cases, the desktop outperformed the laptop in runtimes due to the slightly higher processor speed.

Figure 4 shows the efficiency coefficient, *c*, which is the relative performance increase of adding additional processors compared to the single core case. For a configuration with N cores and run time, t, this value is obtained using Eq. 1:

$$c = \frac{t_{1\_core}}{N(t_{N\_core})} \quad (1)$$

The data suggests that performance gains diminish when increasing from two to four cores on both the laptop on the desktop. This is because as run times decrease from the additional cores, the overhead time associated with creating all of the tasks becomes more significant. For the laptop single, double, and quad core cases, generating the iterations consumed, on average, 1.5%, 2.7%, and 5.1%, respectively, of the total run time.

The data also suggests that the laptop tends to perform worse as the iteration count increases, while the desktop performs relatively better. The most probable cause of this is host recycling. The task per host count was set to 256 iterations. The recycling time of a laptop host averaged sixteen seconds, while host recycling on the desktop took six seconds.

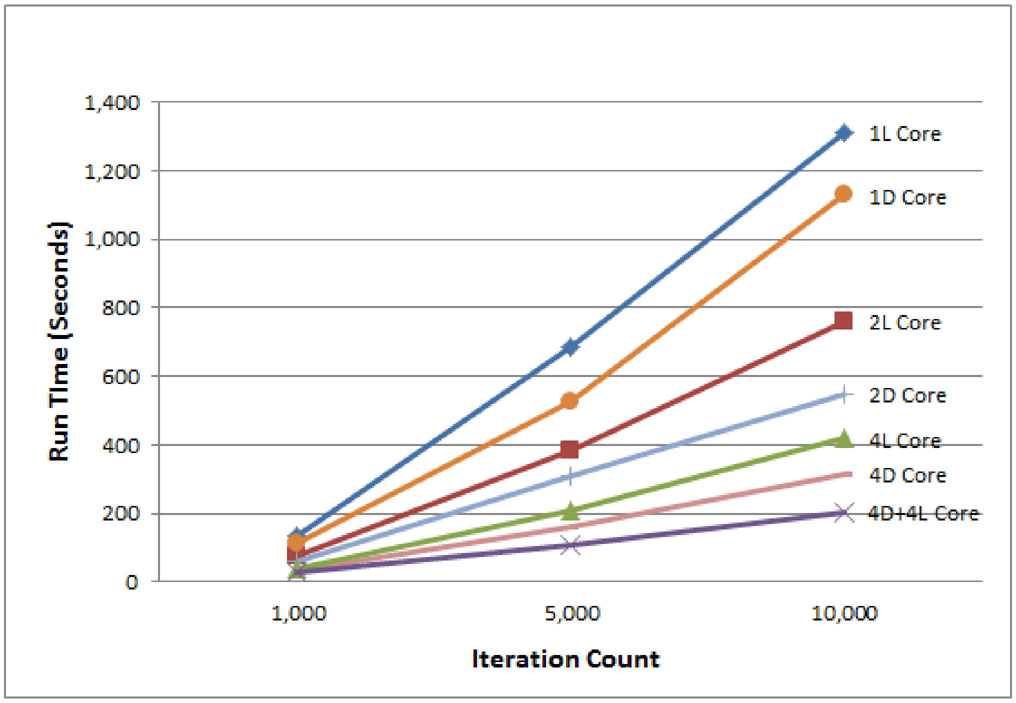

Figure 3. Run Time vs Iteration Count for Monte Carlo Configurations

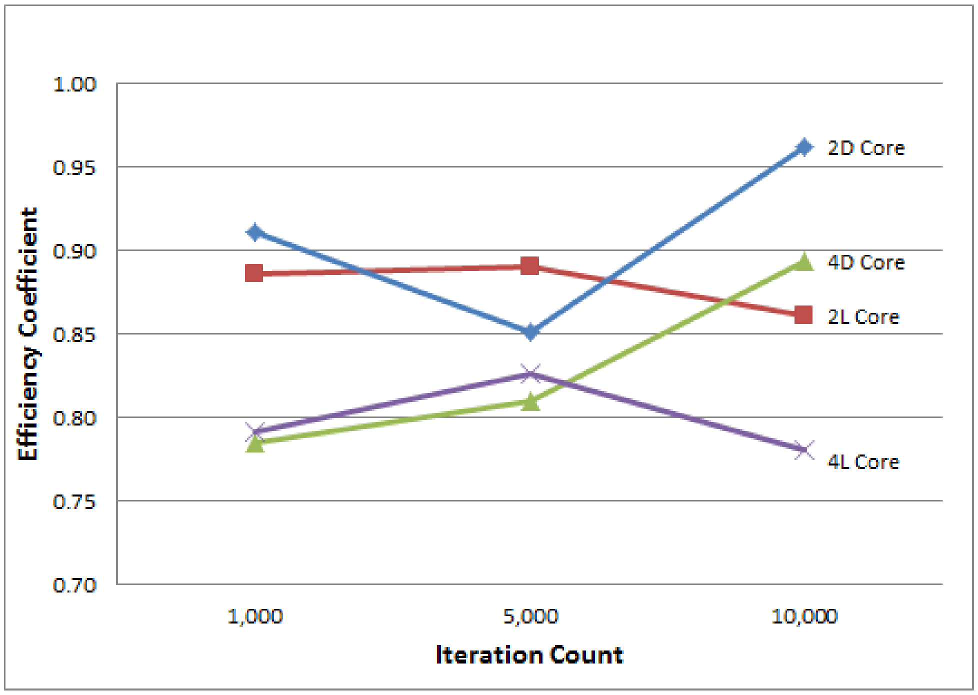

Figure 4. Efficiency Coefficient vs Iteration Count for Monte Carlo Configurations

3. Astrogator Search Plugin Implementation

We next explored an implementation of STK Server which would provide a somewhat more generic and natural extension of the existing Astrogator workflow. In this example an Astrogator Search Profile Plugin was created which uses a simple parametric scan optimization algorithm to evaluate the design space. The plugin has two modes of operation: the first uses the normal single-threaded MCS evaluation available through the plugin interface while the second uses STK Server to run multiple evaluations at one time across all active Agents. The end result is an extension to the Astrogator capabilities and user interface that is nearly transparent to the end user in both the GUI and when automating Astrogator through the Connect and Object Model API.

The STK Server architecture required creating three different libraries to achieve this capability:

1. UI Plugin - The UI plugin is necessary for two main purposes. Similar to the Monte Carlo example, the first is to save the current scenario to a location accessible to all agents. The second is to properly configure the MCS for execution on STK Server, and then begin the execution.

2. Search Profile Plugin - The search profile plugin captures the user inputs for the selected control variables (bounds and step size) and the selected results to be minimized or maximized. It then creates a set of custom STK Server tasks, one for each combination of the control variables. These tasks are packaged into an STK Server job, and the job is submitted to the server. The executes each task and returns the completed jobs with their results, which are then parsed for the optimum value. This value is returned through the user through the Search Profile Plugin API into a custom Astrogator status grid. After completion, the rest of the MCS continues.

3. STK Server task - The STK Server task loads a local copy of the scenario saved by the UI plugin, modifies the MCS controls with the values passed by the Search Profile Plugin, executes the MCS, and returns the result values requested by the Search Profile Plugin.

A simple process diagram is included in Fig. 5.

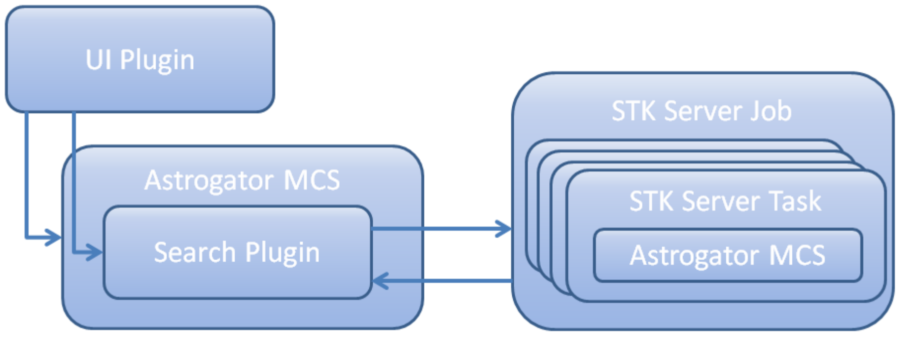

Figure 5. Architecture of STK Server Astrogator Search Plugin

This configuration was tested with a simple MCS which propagates a default Astrogator satellite in a circular low Earth orbit, applies a specified ΔV in the velocity direction, and propagates to apoapsis where the radius from Earth is evaluated. This was setup as a simple optimization problem with ΔV as the single control, where the radius was to be maximized.

A number of cases were evaluated using the agents listed in Tab. 2. Each agent ran STK 10.1 on Windows 7. Note that for all cases, all cores from Laptop 1 were utilized before adding any from Laptop 2. Laptop 2 cores were hyperthreaded and could support 2 hosts per core, for a total of 16 available hosts.

Table 2. Summary of Agent Specifications for Search Profile Application

	Physical Cores	CPU Speed (GHz)	RAM (GB)
Laptop 1	8	2.7	24
Laptop 2	4	2.5	8

The first test case was for a run that utilized all 16 available hosts, but varied the number of iterations required. Figure 6 illustrates these results.

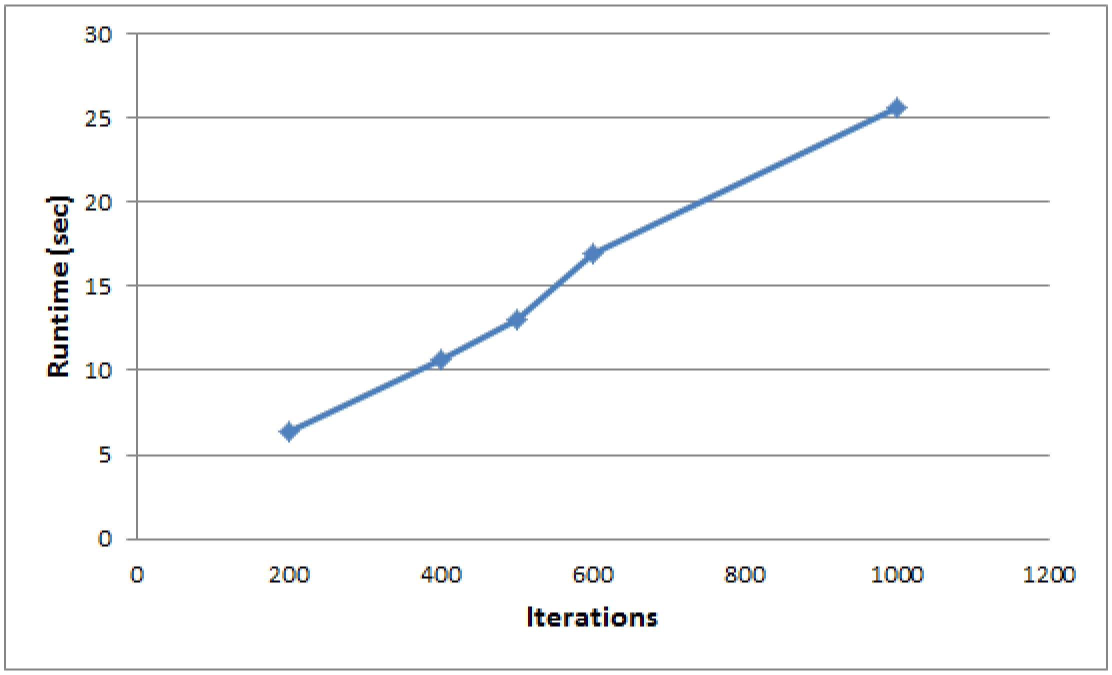

Figure 6. Run Time vs Iterations for Search Plugin Using 16 Hosts

Each data point represents the average of several runs. Note that because of the STK Server settings used, each host would recycle after 256 iterations. When recycling in this test case, a given host is unavailable for around 30 seconds, representing a significant fraction of the overall runtime for this test case. The timing of these recycling events is managed by the STK Server coordinator and may occur at any time during a given run. The data above is for those test cases where no significant host recycling happened. When including host recycling, the trends are not as smooth or predictable and individual runs were observed to be longer by up to the full host recycling duration. Therefore, in cases with short function evaluations, is it recommended to investigate the most appropriate host recycling iteration count.

The second test case evaluated the runtime for a test case of 1000 runs using various numbers of available hosts. The results are shown in Fig. 7. Note that the scale is logarithmic. The horizontal line represents the runtime when using the single-threaded version of the Search Profile Plugin which does not utilize STK Server. The computational overhead required for using the STK Server architecture is clearly seen for the lower host counts, with the breakeven point coming at between 2 and 3 hosts. Note that hosts 8-16 in these runs are from Laptop 2, which offer decreasing gains partly due to lower clock speed as well as adding only hyperthreading for hosts 13-16.

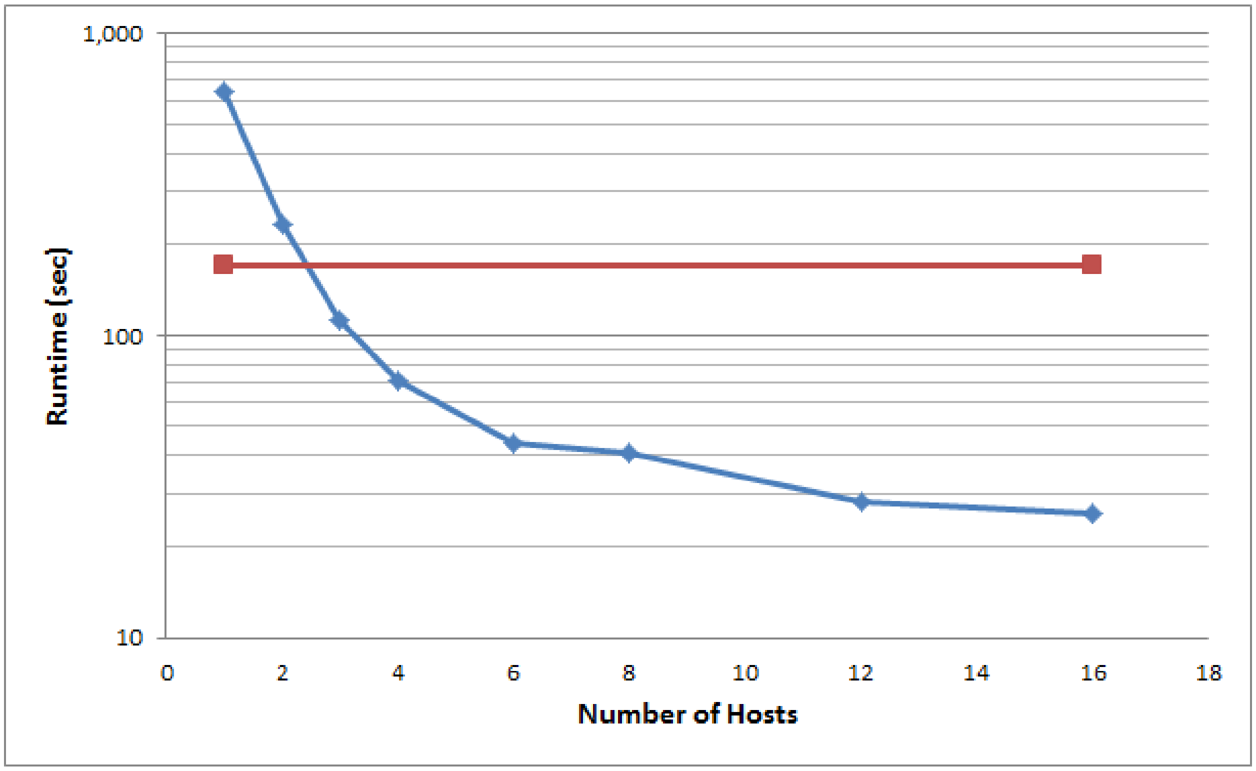

Figure 7. Search Profile Plugin Runtime vs Number of Hosts for 1000 Iterations.

4. Conclusions

STK Server provides a useful architecture for scalable, server-based orbit propagation and trajectory design computations. Combined with the existing APIs in STK, a variety of applications can be significantly enhanced using commodity desktop or laptop hardware. As with other distributed processes, the performance gains scale well with additional hosts. The computational overhead required means that runs with many iterations benefit somewhat more than those with relatively fewer runs. However, even problems with fewer iterations stand to benefit greatly, especially when each individual run is time-consuming.

The flexibility of the STK Server task definition opens a wide array of mission design problems that can utilize STK Server. Any kind of propagation or computation that can be solved using the API can be made into a Server task. This includes additional user-defined plugins, such as force models, engine models, atmospheric models, and much more.

The benefit of parallelizing the default search algorithms in STK Astrogator has already been considered and is the subject of future work [2]. STK Server provides a useful platform for these enhancements. However, it is just as suitable for the extension of other search algorithms. It is noted that algorithms which typically produce a greater number of independent function evaluations (e.g. parameter scans or genetic algorithms) may stand more to gain in performance. However, STK Server makes it more practical to use smaller step sizes or longer function evaluations with any algorithm.

5. References

[1] Carrico, John, and Fletcher, Emmet. Software Architecture and Use of Satellite Tool Kit's Astrogator Module for Libration Point Orbit Missions. 2004. https://www.agi.com/whitepapers.

[2] Berry, Matthew. Comparisons between Newton-Raphson and Broyden's Methods for Trajectory Design Problems. Girdwood, AK: AAS/AIAA Astrodynamics Specialist Conference, 2011.