Gush User Study 1

Jeannie Albrecht

October 2008

This document describes the preliminary user study that I completed to gain an understanding of what aspects of Gush need improvement. I obtained feedback from two different sets of users: an advanced group and a novice group. The advanced group of users was mostly comprised of graduate student systems researchers at UC San Diego. The majority of these users already had significant familiarity with wide-area distributed environments, and most used the terminal and XML-RPC interfaces rather than the GUI. They used Gush to automate tasks associated with running distributed applications on PlanetLab and on ModelNet in the context of their research. This advanced-user study was completed during Spring 2007 and Fall 2007. The novice group of users was comprised of undergraduates at Williams College. These users had little to no experience with distributed programming or wide-area environments. These students used the Gush GUI (called Nebula) to complete a final project in an upper-division Distributed Systems course. The goal of the project was to build and evaluate a peer-to-peer system on PlanetLab using at least 25 machines spread around the world. This novice-user study was completed during Spring 2008. Coincidentally, it is important to note that the novice-user study was performed in close proximity to the OSDI deadline, which meant that PlanetLab hosts were more over-loaded and experienced higher failures than usual.

Advanced-User Study

In this section I summarize the feedback obtained from the advanced users of Gush. These observations were gathered using a survey that was sent to all known Gush users at UC San Diego. Overall, the advanced users were satisfied with the functionality of the Gush terminal and XML-RPC interfaces. When no failures or errors occurred, Gush worked well and simplified the tasks involved with software distribution and application control. However, in the presence of an error or failure, it was difficult to determine the source of the problem using Gush. Moving forward, the biggest area for improvement is error detection and reporting. Gush needs to provide advanced users with mechanisms for efficiently finding bugs in their system. In particular, it is important to make Gush error messages more descriptive and informative so that users can track down problems easier.

The following list highlights some of the feedback obtained from advanced users:

- (Terminal interface) "What I like about [Gush] is that it's flexible enough to work across many administrative domains (something that typical scripts can't always do). It's fairly easy to get installed and setup on a new machine. The structure of the experiments file largely makes sense and is easy to modify/adapt.

  When things go wrong with the experiment, it's often difficult to figure out what happened. The debug output occasionally doesn't include the source of the problem. Overall, it's an extremely useful (if finicky) tool.

- (XML-RPC interface) "I used Gush to orchestrate workflows of batch tasks. Gush made it really easy to insert arbitrary tasks in the workflow. I used Gush's wget mechanisms for doing data staging for scientific apps, and that was a very useful feature as well. There were a few minor problems with the initial setup, but many of them were related to machine configuration and such. Overall the system was solid with few failures."

- (XML-RPC interface) "We began to use Gush as a replacement for gexec in our cluster experiments because we needed a more secure remote execution tool. The transition has been both a blessing and a curse. The version of Gush we use does not yet scale to the number of nodes we would like to use[1], and synchronized start is not as synchronized as we would like due to that scalability issue. However, in exchange for these two headaches we were able to re-vamp our cluster experiment infrastructure with one which does not require privileged support, and what's more, it has the ability to control a multi-phase execution through the use of barriers, which our prior scripts did in an unsafe way (our prior scripts did nothing in terms of detecting failures in phases and aborting or restarting experiments)."

  For reference, we also tested our cluster execution tool with simple parallel ssh. Compared to this, Gush provides extra functionality such as barrier start with a minimal amount of extra effort. However, the synchronized start was slightly better with our parallel ssh because without the ability to use barriers, we wrote scripts which assumed synchronized clocks and performed accordingly. It is noteworthy that this same solution could be applied using Gush, but has not been since Gush supports the barrier start internally."

Novice-User Study

In this section I summarize the feedback obtained from novice Gush users. These users were primarily undergraduates enrolled in my Distributed Systems course at Williams College last spring. The class consisted of 14 students in total, ranging from sophomores to seniors. The students had varying levels of programming expertise, although none had any prior experience with the design and implementation of distributed systems. I conducted the survey by observing the students using Gush during lab sessions. The students worked in pairs, so I was easily able to monitor the progress of 7 groups. Overall, the students were eventually able to use Nebula (the Gush GUI) to run their code on PlanetLab. However, since Nebula is not as mature as the terminal interface, there are still several features that are not supported. In addition, the students had little prior experience with system administration and script writing. Thus when things went wrong, which they inevitably did, Nebula did not provide them with an adequate way to debug their system and diagnose the problem.

The following lists summarizes the key observations made regarding the functionality of the Gush GUI:

- Gush configuration: Getting Nebula and Gush configured and working correctly was a little challenging for some students. This could easily be remedied by adding some default parameters that do not have to be specified except in atypical usage scenarios . For example, the default home directory should be set as $/home/slice\_name$ rather than having them enter this value. Many of them entered their local working directory instead of their home directory on PlanetLab, which meant that Gush and Nebula could not communicate correctly.

[1] This particular user was trying to run on 10,000 virtual ModelNet hosts, and Gush ran out of file descriptors. I believe that the scalability problem has been addressed in the latest release.

- Slice configuration: Adding and remove nodes from slices worked very well. The students were quick to master this feature and thought it was "cool" to pick nodes from a map rather than from a list on a webpage. They pointed out that PlanetLab hostnames were not always descriptive enough for them to discern the node's physical location, but seeing dots on a map made it very clear.

- Application Specification: The students had only small problems specifying applications (e.g., building the XML using the GUI). Most of their applications were simple and did not have special synchronization requirements. There was some minor confusion regarding software package specifications, but it will be easy to add additional details to the software GUI component for clarification.

- Resource Selection: Choosing a usable set of resources was arguably the biggest problem that the students had. They all proceeded in basically the same way, and all eventually encountered the same major stumbling block. They started by adding all available machines to their slice. Then they specified their application, without using SWORD[2]. The Gush matcher would randomly choose resources that were either not functioning or experienced high delays (for example, Chinese nodes have low bandwidth connections in general) and something would go wrong. They then removed most of the nodes from their slice, and tried to force the matcher to always choose a certain set of nodes. This worked until one of the chosen nodes failed, and then the matcher was unable to find a replacement, causing the experiment to fail without recovery. In the terminal version of Gush, there is a way to specify a preference file that helps the matcher choose "good" nodes without using SWORD. This feature is not available in the GUI. Moving forward, I think we need to significantly improve the resource matching and selection functionality in Gush.

- Software Installation: Gush currently does not integrate any package management functionality from external services such as Stork/Raven. As a result, Gush requires users to manually install common packages (such as Java). For many students, this was problematic. Automating the installation of Java is difficult, and using yum to install software on PlanetLab machines is unreliable at best. In addition, different versions of some libraries (like Python) caused strange bugs that my students had a hard time overcoming. I am not sure if this is something that Gush should support, but it would be helpful to provide GENI users with a way to reliably configure their execution environment with the same versions of software packages.

- Experiment Control: Once my students were finally able to configure a stable set of machines, controlling experiments using the Gush GUI went reasonably well. When their code worked, Gush was easily able to stop and start their executions. However when their code failed, there were some failure cases from which Gush was unable to recover. In some extreme cases, the students ended up with several different versions of their code running simultaneously on PlanetLab, and cleaning up old processes is not something that Gush currently supports. It should be noted that the students who encountered the most problems were also the ones with the least prior experience, but ultimately I hope that Gush will support even the most novice users.

[2]SWORD is a service that performs application-specific resource discovery on PlanetLab (http://sword.cs.williams.edu).