Managing Distributed Applications using Gush

Jeannie Albrecht and Danny Y. Huang
Williams College

http://gush.cs.williams.edu
Overview

• How do experimenters actually use GENI?
• Goal: Develop abstractions and tools for addressing the challenges of managing distributed applications
  • Make it easy for a range of users (including students!) to run a variety of experiments on GENI

• Strategy
  • Make minimal assumptions about GENI “resources” and how they are allocated
  • Leverage existence of lower level services to locate resources and obtain credentials
  • Interface with other user tools
  • Hide complexity and use one common user interface to interact with different underlying systems (i.e., PlanetLab, ProtoGENI/Emulab, ORCA)
Gush

- A distributed application management infrastructure
  - Designed to simplify deployment of distributed applications
  - Provides abstractions for configuration and management
  - Allows users to “remotely control” computers running distributed applications
Step 1: Describe Application

- Describe experiment using application “building blocks”
- Create customized control flow for distributed applications
- **Application specification** blocks are described using XML

![Diagram of application blocks and processes](image-url)
Step 1: App Spec

```xml
<gush>
  <project name="simple">
    <software name="SimpleSoftwareName" type="none">
      <package name="Package" type="web">
        <path>http://sysnet.cs.williams.edu/~jeannie/software.tar</path>
        <dest_path>software.tar</dest_path>
      </package>
    </software>
    <component name="Cluster1">
      <rspec>
        <num_hosts>20</num_hosts>
      </rspec>
      <software name="SimpleSoftwareName" />
      <resources>
        <resource type="planetlab" group="williams_gush" />
        <resource type="gpeni" group="gpeni_gush" />
        <resource type="max" group="maxpl_gush" />
      </resources>
    </component>
    <experiment name="simple">
      <execution>
        <component_block name="cb1">
          <component name="Cluster1" />
          <process_block name="p2">
            <process name="cat">
              <path>cat</path>
              <cmdline>
                <arg>software.txt</arg>
              </cmdline>
            </process>
          </process_block>
        </component_block>
      </execution>
    </experiment>
  </project>
</gush>
```
Step 1: App Spec

```xml
<gush>
  <project name="simple">
    ...
    <component name="Cluster1">
      <rspec>
        <num_hosts>20</num_hosts>
      </rspec>
      <software name="SimpleSoftwareName" />
      <resources>
        <resource type="planetlab" group="williams_gush" />
        <resource type="gpeni" group="gpeni_gush" />
        <resource type="max" group="maxpl_gush" />
      </resources>
    </component>
    <component name="Cluster2">
      <rspec>
        <num_hosts>20</num_hosts>
        <orca>
          <num_hosts>20</num_hosts>
          <type>1</type>
          <memory>784</memory>
          <bandwidth>300</bandwidth>
          <cpu>75</cpu>
          <lease_length>12000</lease_length>
          <server>http://geni.renci.org/orca:8080</server>
        </orca>
      </rspec>
      <software name="SimpleSoftwareName" />
      <resources>
        <resource type="ssh" group="orca" />
      </resources>
    </component>
    ...
  </project>
</gush>
```

• Application level control framework interoperability in GENI!
Step 2: Acquire Resources

- How can we find “good” resources?
  - We may want machines with specific characteristics
- Gush interfaces directly with lower level services
  - Gush fully supports PlanetLab resources
  - Beta support for ORCA and ProtoGENI resources
<gush>
  <resource_manager type="geni">
    <user>plc.williams.jeannie</user>
    <config_file>planetlab_sfi_config</config_file>
    <port_map slice="plc.williams.gush" port="15413"/>
  </resource_manager>

  <resource_manager type="geni">
    <user>plc.ksu.jeannie</user>
    <config_file>gpeni_sfi_config</config_file>
    <port_map slice="plc.ksu.gush" port="15414"/>
  </resource_manager>

  <resource_manager type="geni">
    <user>plc.max.jeannie</user>
    <config_file>max_sfi_config</config_file>
    <port_map slice="plc.max.gush" port="15415"/>
  </resource_manager>
</gush>
ORCA Resource Specification

• Unlike PlanetLab, ORCA resources do not exist in advance
• ORCA creates VMs on demand and emphasizes resource isolation
• ORCA resources are defined within application specification
ORCA Resource Specification

```xml
<component name="VMGroup1">
  <rspec>
    <num hosts>20</num hosts>
    <orca>
      <num hosts>20</num hosts>
      <type>1</type>
      <memory>784</memory>
      <bandwidth>300</bandwidth>
      <cpu>75</cpu>
      <lease length>12000</lease length>
      <server>http://geni.renci.org/orca:8080</server>
    </orca>
  </rspec>
  <resources>
    <resource type="ssh" group="orca"/>
  </resources>
</component>
```

- Gush contacts ORCA slice manager when experiment is started
- ORCA calls back to Gush when resources are ready for use
ProtoGENI Resource Directory

• ProtoGENI resources are defined like PlanetLab resources
• Experiments must be swapped in and out before execution
• Like ORCA, ProtoGENI resources are created on demand
• Unlike ORCA, ProtoGENI currently does not provide callbacks to Gush about resource availability
Step 3: Configure Resources

- Connect to and configure selected resources
  - **Controller** “remotely controls” the **clients** on the experimenter’s behalf
  - Install software on clients
Step 4: Start Application

- Controller issues commands to clients telling them to start running applications/experiments
  - Senders begin running sender processes
  - Receivers begin running receiver processes
We want to make sure the processes keep running.

Clients monitor experiment processes for failures:
- If a failure is detected, client notifies controller.
- Controller decides to tell client to restart failed process.
Step 6: Cleanup

- Gush clients make sure all programs exited cleanly
- Remove logs and software from remote machines
- Disconnect clients from controller
Gush in Action

gush> load ./tests/simple.xml
Project "simple" is selected.
Experiment "simple" is selected.
gush> run
Starting experiment run.
Running experiment simple...
gush> The configuration matcher has finished matching.
The resource allocator has finished successfully.
gpeni_gush@geni-planetlab-1.ksu.gpeni.net:15414 has joined the mesh.
The file transfer of Package to geni-planetlab-1.ksu.gpeni.net has been completed.
The software installation of Package on geni-planetlab-1.ksu.gpeni.net was successful.
williams_gush@planetlab1.williams.edu:15413 has joined the mesh.
maxpl_gush@planetlab2.dragon.maxgigapop.net:15415 has joined the mesh.
The file transfer of Package to planetlab1.williams.edu has been completed.
The software installation of Package on planetlab1.williams.edu was successful.
The file transfer of Package to planetlab2.dragon.maxgigapop.net has been completed.
The software installation of Package on planetlab2.dragon.maxgigapop.net was successful.
gpeni_gush@geni-planetlab-1.ksu.gpeni.net:15414,31821: Hello World
williams_gush@planetlab1.williams.edu:15413,19548: Hello World
maxpl_gush@planetlab2.dragon.maxgigapop.net:15415,26459: Hello World
The experiment has ended.
• Nebula (GUI) allows users to describe, run, monitor, & visualize applications
• XML-RPC interface for managing applications programatically
Future Work and Conclusions

• 18 undergrads at Williams College used Gush and Nebula to run experiments on PlanetLab last fall
  • Gush was stable, Nebula needs work
  • iPod/iPhone interface?
  • 2 undergrads have worked on Gush development
  • 2 more will work on Nebula this summer
• Need better support for wireless/mobile devices
• Gush is probably not the solution for all testbeds
  • But it’s a step in the right direction (I hope)!
• Gush has helped identify what users actually want and need
  • Determine the right set of abstractions for experiment management and application control
Thanks!

For more info:
http://gush.cs.williams.edu

Gush