Automatically Establishing the Execution Environment for User Applications

Paul Kolano

AMTI/NASA Ames Research Center

kolano@nas.nasa.gov

Overview of Presentation

- Background
- Establishing Execution Environments
- Implementation Details
- Conclusions
- Future Work

The Problem

- A major goal of grid computing is to transparently run applications across different resources
- Different resources may have different setups both within and between organizations
  - Different software installed
  - Different file system structures
  - Different default environment settings

The Problem (cont.)

- Running applications on new resources typically results in:
  - someexec: not found
  - /usr/libexec/ld-elf.so.1: Shared object "somelib.so" not found
  - Exception in thread "main" java.lang.NoClassDefFoundError: some/java/Class
  - Can't locate Some/Perl/Mod.pm in @INC
  - ImportError: No module named some.python.mod

The Problem (cont.)

- Users end up wasting time setting up the resources that were supposed to save them time

The Goal

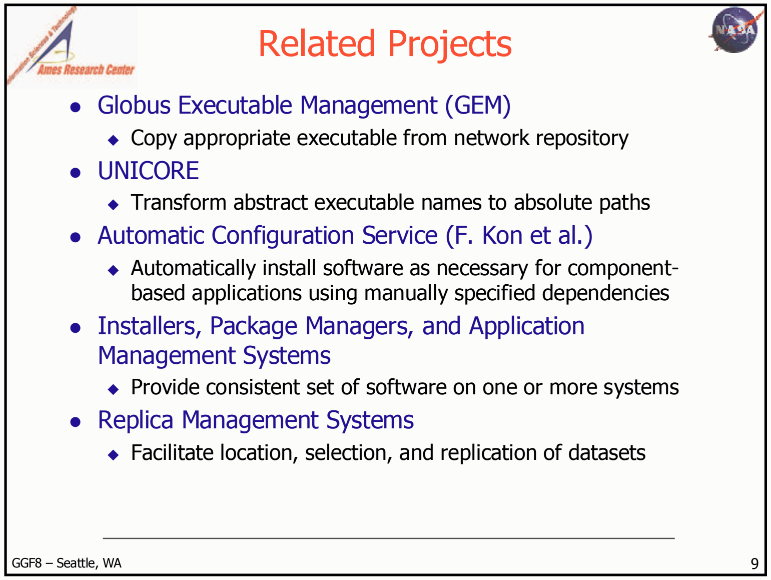
- Automatically ensure that user applications will not encounter software dependency failures during execution

Typical Solutions

- Statically-linked executables
  - Result in
    - Overly large executables
    - Inefficient use of memory
    - Hard-coding library bugs into code
- Custom software packages
  - Require detailed knowledge of
    - Dependency analysis techniques
    - Differences in environment settings
      - Operating systems
      - Software types

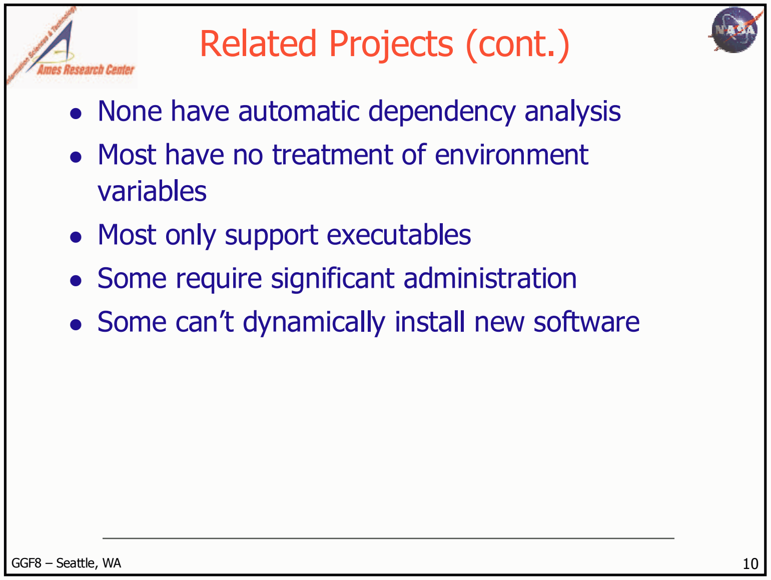
Typical Solutions (cont.)

- Waste time and allocations better spent on actual work
  - Transferring unnecessarily large files
  - Manually preparing custom packages



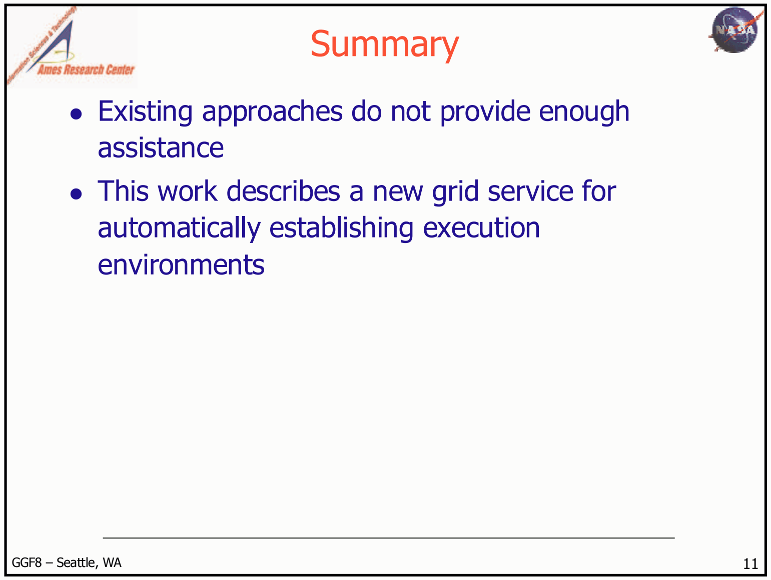
Related Projects

- Globus Executable Management (GEM)
  - Copy appropriate executable from network repository
- UNICORE
  - Transform abstract executable names to absolute paths
- Automatic Configuration Service (F. Kon et al.)
  - Automatically install software as necessary for component-based applications using manually specified dependencies
- Installers, Package Managers, and Application Management Systems
  - Provide consistent set of software on one or more systems
- Replica Management Systems
  - Facilitate location, selection, and replication of datasets



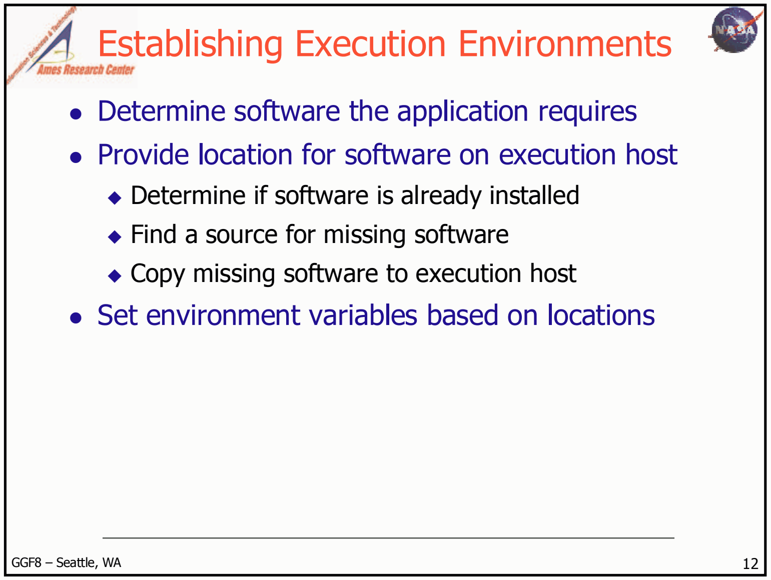
Related Projects (cont.)

- None have automatic dependency analysis
- Most have no treatment of environment variables
- Most only support executables
- Some require significant administration
- Some can't dynamically install new software



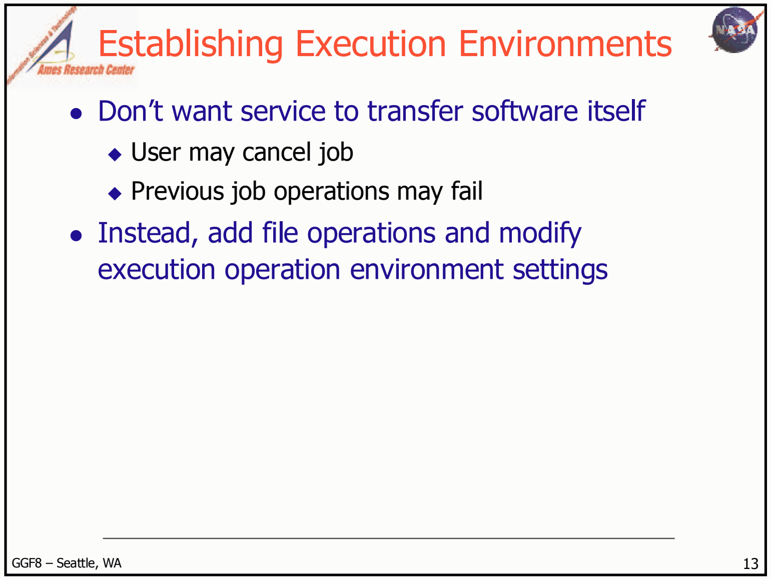
Summary

- Existing approaches do not provide enough assistance
- This work describes a new grid service for automatically establishing execution environments



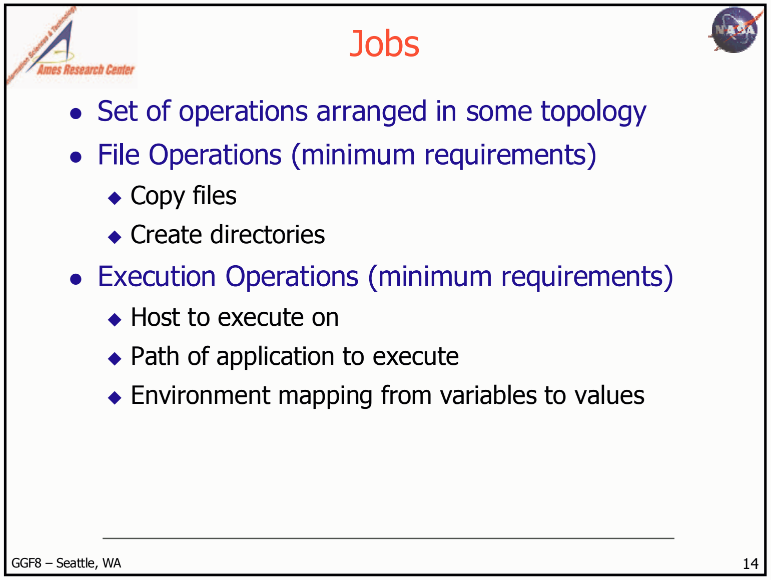
Establishing Execution Environments

- Determine software the application requires
- Provide location for software on execution host
  - Determine if software is already installed
  - Find a source for missing software
  - Copy missing software to execution host
- Set environment variables based on locations



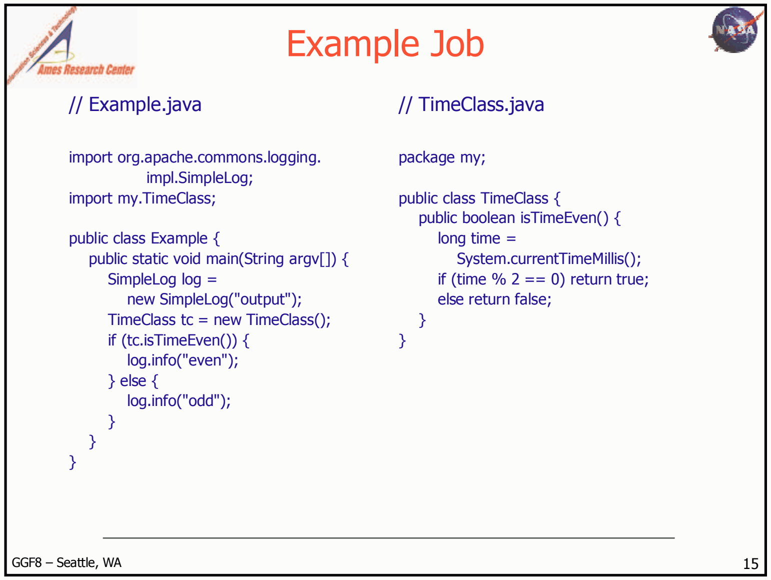
Establishing Execution Environments

- Don’t want service to transfer software itself
  - User may cancel job
  - Previous job operations may fail
- Instead, add file operations and modify execution operation environment settings



Jobs

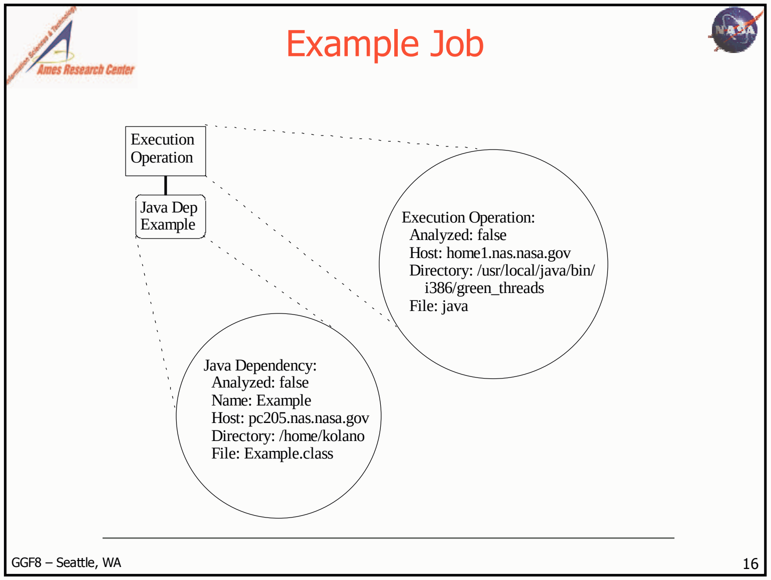
- Set of operations arranged in some topology
- File Operations (minimum requirements)
  - Copy files
  - Create directories
- Execution Operations (minimum requirements)
  - Host to execute on
  - Path of application to execute
  - Environment mapping from variables to values



Example Job

```
// Example.java

import org.apache.commons.logging.
            impl.SimpleLog;
import my.TimeClass;

public class Example {
    public static void main(String argv[]) {
        SimpleLog log =
            new SimpleLog("output");
        TimeClass tc = new TimeClass();
        if (tc.isTimeEven()) {
            log.info("even");
        } else {
            log.info("odd");
        }
    }
}
```

```
// TimeClass.java

package my;

public class TimeClass {
    public boolean isTimeEven() {
        long time =
            System.currentTimeMillis();
        if (time % 2 == 0) return true;
        else return false;
    }
}
```



Example Job

Execution Operation:
Analyzed: false
Host: home1.nas.nasa.gov
Directory: /usr/local/java/bin/i386/green_threads
File: java

Java Dependency:
Analyzed: false
Name: Example
Host: pc205.nas.nasa.gov
Directory: /home/kolano
File: Example.class

Stage 1: Dependency Analysis

- Determine application software dependencies
- Support common software types
- Concentrate initially on statically generated dependencies (i.e. do not worry about cases such as char *lib = f(); dlopen(lib))

Dependencies

- Basic information
  - Type
    - e.g. Executable
  - Name
    - e.g. w3m
  - Version range
    - e.g. [3.0, 3.1.1]
  - Feature list
    - e.g. compiled with SSL support
- Extended information
  - Source host
  - Source path
  - Target path
  - Analyzed flag

Dependencies (cont.)

- Currently supported types
  - Executable and Linking Format (ELF) objects
    - Executables
    - Shared libraries
  - Java classes
  - Perl programs
  - Python programs

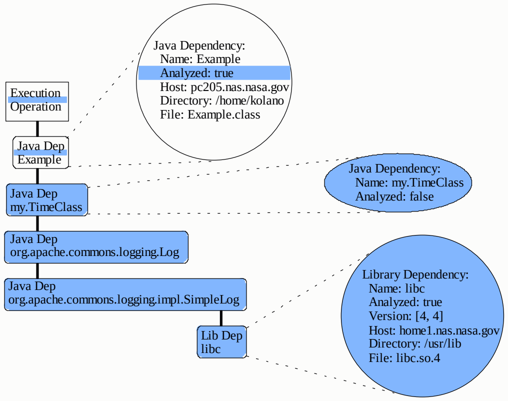
Stage 1: Dependency Analysis (cont.)

- Dependency information is embedded
  - ELF executables and libraries
  - Java classes
- Dependency information must be derived
  - Perl programs
  - Python programs
  - Techniques
    - Textually traverse for relevant expressions
    - Partially evaluate using interpreter mechanisms

Stage 1: Dependency Analysis (cont.)

- Use as many existing tools as possible
  - Executables and libraries
    - ldd, elfdump
  - Java classes
    - com.sun.jini.tool.ClassDep
  - Perl programs
    - Module::ScanDeps
  - Python programs
    - modulefinder

Example Job After Dependency Analysis



Execution Operation

Java Dep Example

Java Dep my.TimeClass

Java Dep org.apache.commons.logging.Log

Java Dep org.apache.commons.logging.impl.SimpleLog

Lib Dep libc

Java Dependency:
Name: Example
Analyzed: true
Host: pc205.nas.nasa.gov
Directory: /home/kolano
File: Example.class

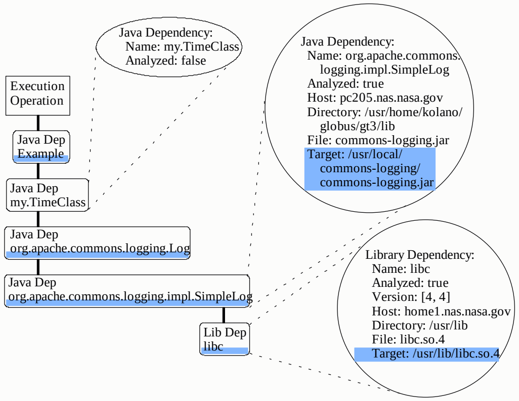
Java Dependency:
Name: my.TimeClass
Analyzed: false

Library Dependency:
Name: libc
Analyzed: true
Version: [4, 4]
Host: home1.nas.nasa.gov
Directory: /usr/lib
File: libc.so.4

Stage 2(a): Dependency Location

- Determine existing location of software on target system
- Minimize number of files to transfer
- Searching an entire file system is impractical
- Must limit search space to specific paths
  - Which paths?
    - Add paths based on Filesystem Hierarchy Standard
    - Add paths based on user and system-default settings
  - Cannot guarantee file will be found in all cases
- Find using ls and Java, Perl, and Python interpreters

Example Job After Dependency Location



Execution Operation

Java Dep Example

Java Dep my.TimeClass

Java Dep org.apache.commons.logging.Log

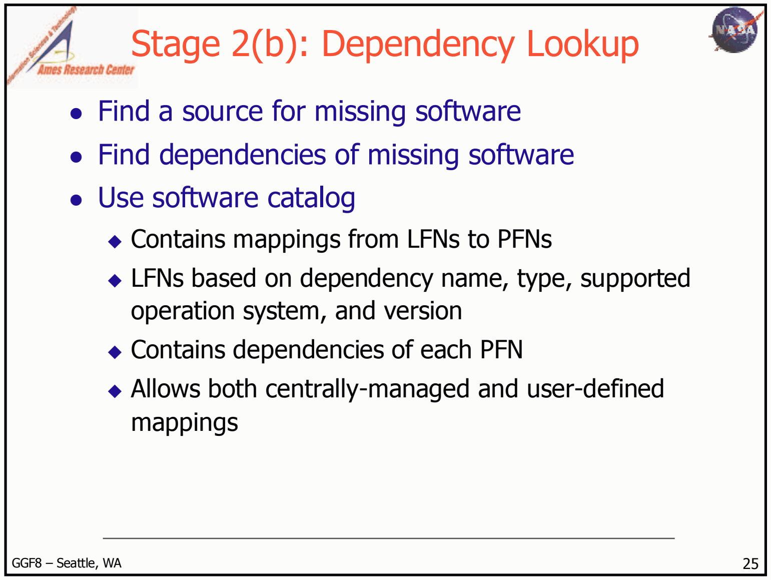
Java Dep org.apache.commons.logging.impl.SimpleLog

Lib Dep libc

Java Dependency:
Name: my.TimeClass
Analyzed: false

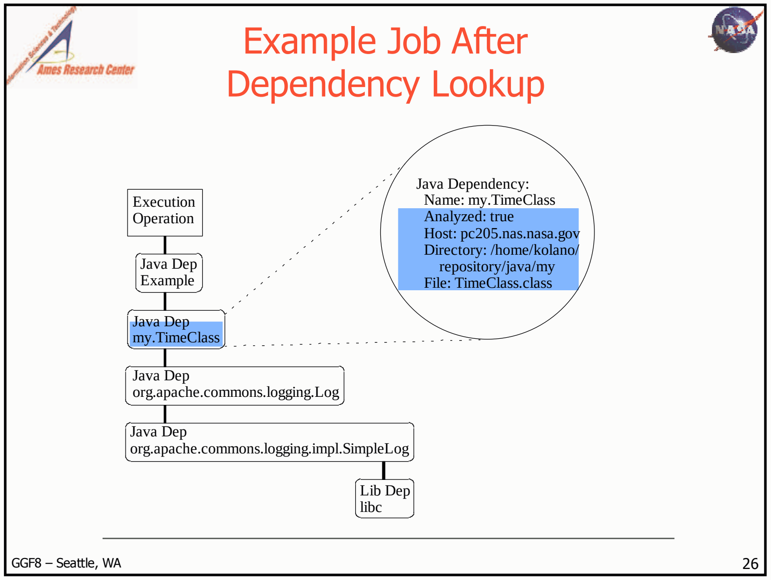
Java Dependency:
Name: org.apache.commons.
logging.impl.SimpleLog
Analyzed: true
Host: pc205.nas.nasa.gov
Directory: /usr/home/kolano/
globus/gt3/lib
File: commons-logging.jar
Target: /usr/local/
commons-logging/
commons-logging.jar

Library Dependency:
Name: libc
Analyzed: true
Version: [4, 4]
Host: home1.nas.nasa.gov
Directory: /usr/lib
File: libc.so.4
Target: /usr/lib/libc.so.4



Stage 2(b): Dependency Lookup

- Find a source for missing software
- Find dependencies of missing software
- Use software catalog
  - Contains mappings from LFNs to PFNs
  - LFNs based on dependency name, type, supported operation system, and version
  - Contains dependencies of each PFN
  - Allows both centrally-managed and user-defined mappings



Example Job After Dependency Lookup

Execution Operation

Java Dep Example

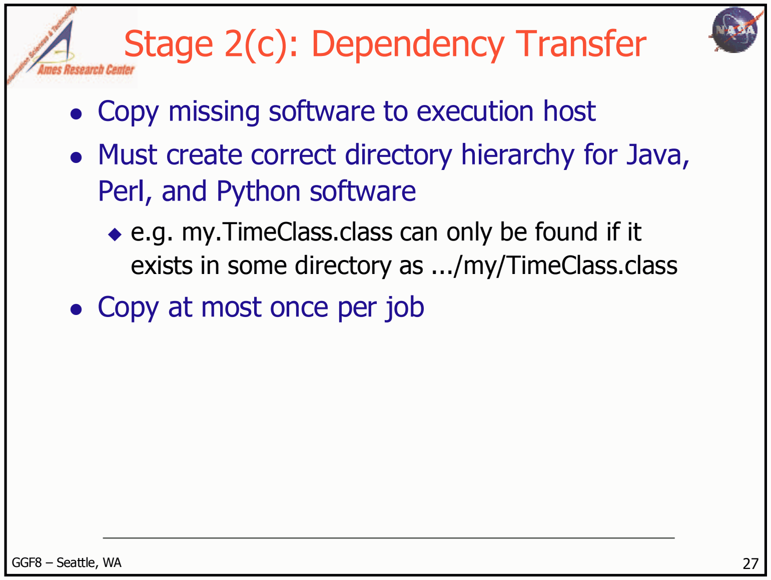
Java Dep my.TimeClass

Java Dep org.apache.commons.logging.Log

Java Dep org.apache.commons.logging.impl.SimpleLog

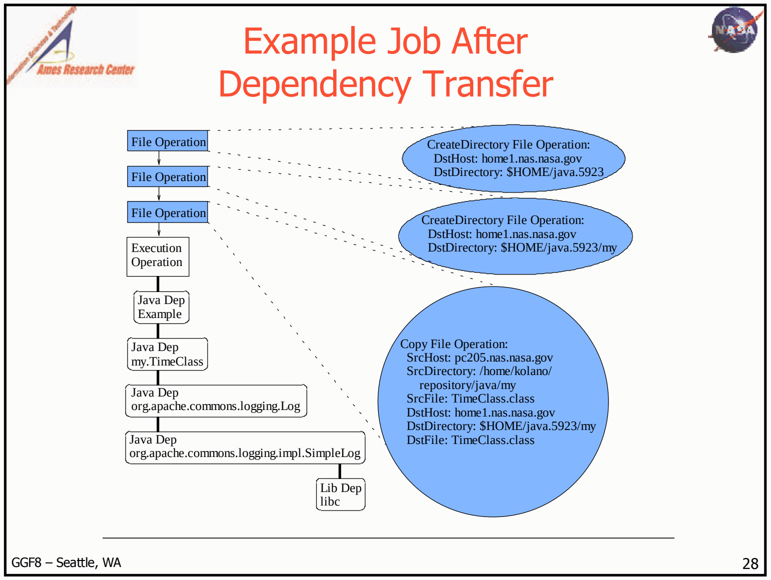
Lib Dep libc

Java Dependency:
Name: my.TimeClass
Analyzed: true
Host: pc205.nas.nasa.gov
Directory: /home/kolano/repository/java/my
File: TimeClass.class



Stage 2(c): Dependency Transfer

- Copy missing software to execution host
- Must create correct directory hierarchy for Java, Perl, and Python software
  - e.g. my.TimeClass.class can only be found if it exists in some directory as .../my/TimeClass.class
- Copy at most once per job



Example Job After Dependency Transfer

File Operation

File Operation

File Operation

Execution Operation

Java Dep Example

Java Dep my.TimeClass

Java Dep org.apache.commons.logging.Log

Java Dep org.apache.commons.logging.impl.SimpleLog

Lib Dep libc

CreateDirectory File Operation:
DstHost: home1.nas.nasa.gov
DstDirectory: $HOME/java.5923

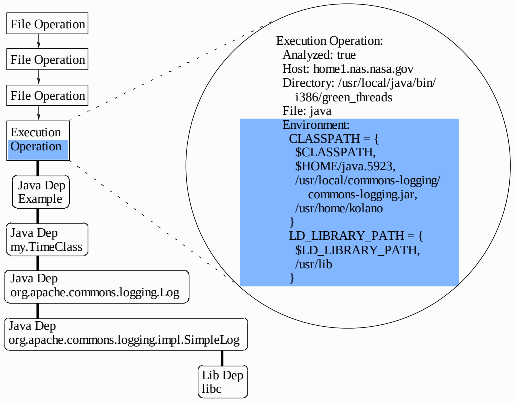
CreateDirectory File Operation:
DstHost: home1.nas.nasa.gov
DstDirectory: $HOME/java.5923/my

Copy File Operation:
SrcHost: pc205.nas.nasa.gov
SrcDirectory: /home/kolano/repository/java/my
SrcFile: TimeClass.class
DstHost: home1.nas.nasa.gov
DstDirectory: $HOME/java.5923/my
DstFile: TimeClass.class

Stage 3: Variable Setup

- Application must be able to locate software
- Set environment variables to find existing and soon to be existing software
- Must consider directory hierarchy for Java, Perl, Python software
  - e.g. for my.TimeClass at /somedir/my/TimeClass.class, CLASSPATH must contain /somedir

Example Job After Variable Setup



```
Execution Operation:
  Analyzed: true
  Host: home1.nas.nasa.gov
  Directory: /usr/local/java/bin/
     i386/green_threads
  File: java
  Environment:
    CLASSPATH = {
      $CLASSPATH,
      $HOME/java.5923,
      /usr/local/commons-logging/
         commons-logging.jar,
      /usr/home/kolano
    }
    LD_LIBRARY_PATH = {
      $LD_LIBRARY_PATH,
      /usr/lib
    }
```

Possible Failures

- Application depends on A, but A cannot be located anywhere
- Application depends on A, which depends on B, but analysis techniques used on A are inadequate to determine B is a dependency
- Application does not depend on A, but analysis techniques report A is a dependency

Dealing with Failures

- Notify user to prevent wasted effort
- Missing software
  - Provide convenience methods to locate relevant dependencies after transformation
- False negatives and false positives
  - Cannot detect automatically
  - Currently, only Perl analysis is susceptible
  - Provide user with flexibility to compensate when necessary

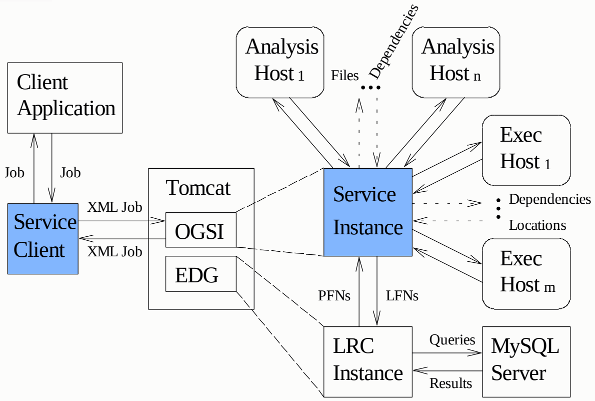
Flexibility

- User has complete control of job transformation
  - Can execute stages individually
  - Can specify dependencies manually
  - Can turn analysis off for individual items
  - Can specify an exact source for software
  - Can specify an existing location on execution host
  - Can manage personal software catalog

Implementation

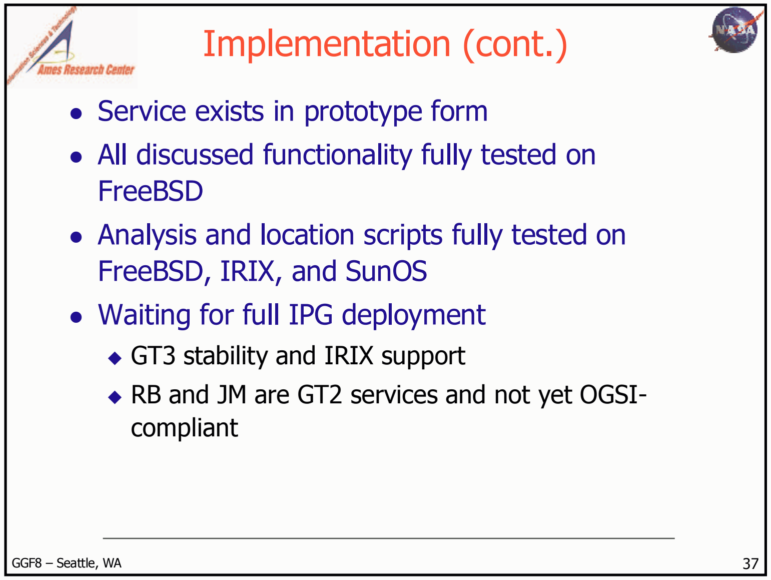
- Implemented in Java and Bourne shell scripts
- Runs as an OGSI-compliant grid service
- Uses OGSI GRAM service to execute analysis and location scripts
- Uses European DataGrid Local Replica Catalog as software catalog

Implementation (cont.)



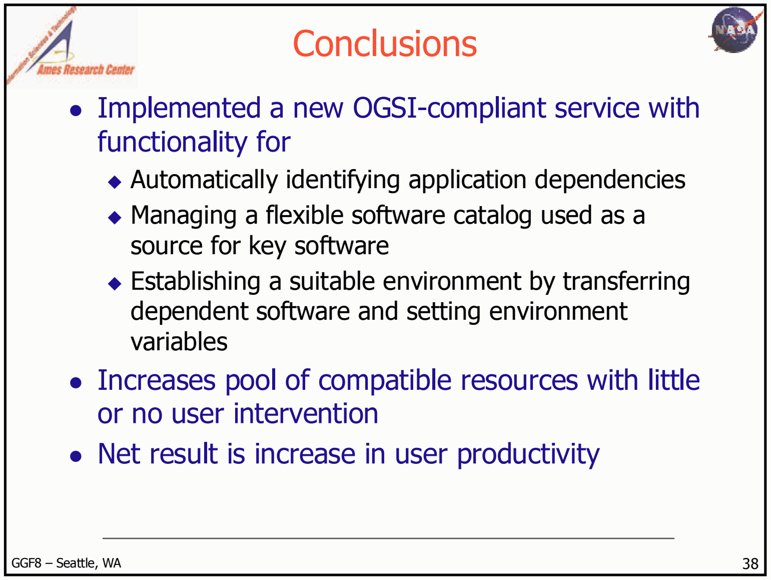
Where does this fit in?

- NASA Information Power Grid (IPG)
  - Current prototype services
    - Resource Broker
      - Select resources for jobs based on user constraints
    - Job Manager
      - Reliably execute jobs on specific resources
  - Establish environment after selection and before execution



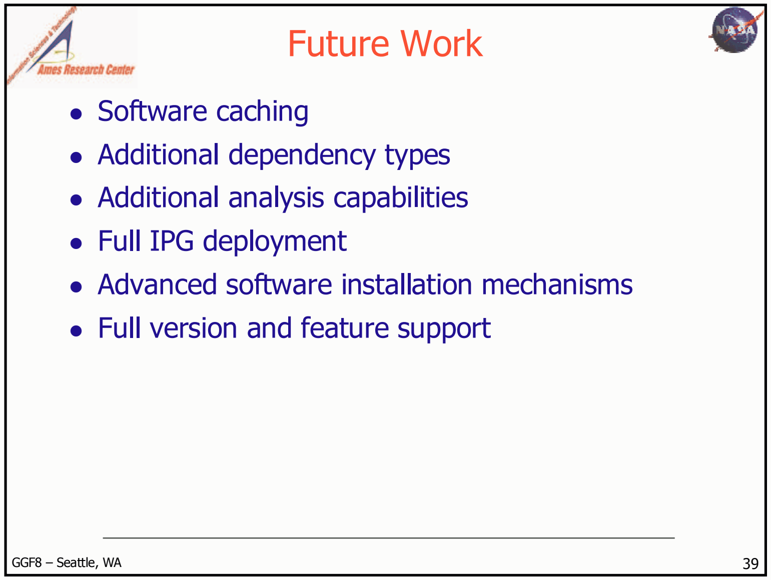
Implementation (cont.)

- Service exists in prototype form
- All discussed functionality fully tested on FreeBSD
- Analysis and location scripts fully tested on FreeBSD, IRIX, and SunOS
- Waiting for full IPG deployment
  - GT3 stability and IRIX support
  - RB and JM are GT2 services and not yet OGSI-compliant



Conclusions

- Implemented a new OGSI-compliant service with functionality for
  - Automatically identifying application dependencies
  - Managing a flexible software catalog used as a source for key software
  - Establishing a suitable environment by transferring dependent software and setting environment variables
- Increases pool of compatible resources with little or no user intervention
- Net result is increase in user productivity



Future Work

- Software caching
- Additional dependency types
- Additional analysis capabilities
- Full IPG deployment
- Advanced software installation mechanisms
- Full version and feature support