Explorations in the Grid/WS Computing Jungle

A Narrative of the Reverse-Engineering Attempts of a “New” Distributed Paradigm

François Taïani, Matti Hiltunen & Rick Schlichting

Opera Talk, 6 May 2008, Comp. Dept., Cambridge University
The best and safest method of philosophizing seems to be, first to inquire diligently into the properties of things, and to establish those properties by experiences and then to proceed more slowly to hypotheses for the explanation of them. For hypotheses should be employed only in explaining the properties of things, but not assumed in determining them; unless so far as they may furnish experiments.

Isaac Newton
Context: Middleware Complexity

ORBacus Request Processing

one request,
2065 individual invocations,
over 50 C-functions and 140 C++ classes.
Web Services’ Bright New World

- **Grid Computing:** federating resources
- **Web Services:** integrating services

- Globus (Argonne Lab.): reference implementation
- Large, complex, collaborative middleware (IBM, Apache,...)

**Very poor performances:**
- Up to 30s to **create** a simple distributed object (counter)
- Up to 2s for a roundtrip remote **add** operation on this counter

**Where does it come from?**
**Does it tell us something about modern mw development?**
Globus

- **Reference Implementation** of the Grid Standards.
  - Developed by the “Globus alliance”, a partnership around Argonne National Laboratory.

- Globus is a **high level “middleware”** (software glue)
  - It offers services to share/ use remote distributed “resources” (CPU, memory, DB, bandwidth)

- Since version 3.9.x use **Web Service “connectivity”**
  - Web Services: “connectivity” middleware across the Internet
  - Integration of services across organizations
  - Related alphabet soup: SOAP, XML, WSDL, HTTP, .NET, etc.
Exploration Goals

- We wanted to **understand** Globus (at least its connectivity)

- Huge piece of software (3.9.x):
  - 123,839 lines in Java (without reused libraries)
  - 1,908,810 lines in C/C++ (including reused libraries)

- Many libraries / technologies involved:
  - XML, WSDL (Descr. Lang), WSRF (Resource Framework)
  - Axis (Java to SOAP), Xerces (XML Parsing), com.ibm.wsdl

- **How to understand that?**

- A typical **reverse engineering** problem
Methodology I + First Results

- First attempt: tracing everything (outside the JVM libs)
  - client: \(1,544,734\) local method call (sic)
  - server: \(6,466,652\) local method calls (sic) [+time out]

- How to visualize such results?
Program Visualization: a few Notions

- Problem studied for quite a long time now.
- Different aspect: collection, manipulation, visualization.
- Visualization some form of projection (many proposed).
- Our goal: understand software structure:

```
lib 1
lib1.Wale .breath
lib1.Mammal.inhale

lib 2
lib2.Lung .inhale
lib2.Muscle.contract
lib2.Nerve .transmit

lib 3
lib3.Signal.travel
```

Growing Call Stack

- Tracing calls reveals the software structure.
Methodology I

lib1.Wale .breath
lib1.Mammal.inhale
lib2.Lung .inhale
lib2.Muscle.contract
lib2.Nerve .transmit
lib3.Signal.travel

A call graph obtained by tracing

⇒ Aggregates invocations of the same library.
⇒ Chart w.r.t. position in call stack.
Methodology I

Package Activity vs. Stack Depth

Software Structure
Package Activity vs. Stack Depth

(client, 1 creation, 4 requests, 1 destruction)
Package Activity vs. Stack Depth

(client, 1 creation, 4 requests, 1 destruction)
Package Activity vs. Stack Depth

(client, 1 creation, 4 requests, 1 destruction)

Looks better, but is the same!

89% of invocations (1,372,534) due to XML!
Package Activity vs. Stack Depth

(client, percentage view)
XML used by org.apache.axis, not by Globus!

Very long stack → probably recursive parsing!
What does it tell us?

- Most of the **local invocations** (89%) are related to **XML** parsing (org.apache.xerces, org.apache.xml).

- The parsing is used by **Axis** (the SOAP/Java bridge from the apache foundation), not directly by Globus.

- The **very long stacks** we observe (up to 57 frames!) most probably reflect some **recursive parsing loop**, rather than the program structure.

- **Similar** findings on the **server** side (only more dramatic, stack depth up to 108 (sic), 4 times more invocations).
New Questions

More insight needed:

- Does invocation count reflect real performance?
- How “bad” is really the platform?
- Can we do the same kind of “structural” projection of profiling data?
- If yes, is it useful?

Our choice: 2 step approach

- (1) Black box profiling
- (2) “Internal” profiling using sampling
Chosen Approach

- 2 steps:
  1. **Black box profiling**: minimal interferences. Coarse results.
  2. **Sample based profiling**: less accurate but more detailed.

- We focused on the **connectivity** of the **WSRF** implementation of GT4-Java:
  - Low level “**plumbing**”. No high level service involved
  - Motivation: profile the **founding bricks** of the Globus platform

- Experimental set-up:
  - **Standalone SMP** server running 4 Intel Xeon @ 1.6GHz
  - **No network cost** involved!
  - **Avoids context switching** overhead!
  - Globus **3.9.4** used (last GT4 alpha release, released Dec.04)
Black-Box Profiling: Approach

- Black Box Approach: Measure **externally** visible latencies
  - Many different situations to be considered!

![Diagram showing the influence of resource init, client init, and container init on latencies.](image)

- Influence of resource init: $\times 5$
- Influence of client init: $\times 5$
- Influence of container init: $\times 5$

Averaging across 10,000 invocations: $\times 50$
Resource Set-Up

Container init overhead (~8.2s!)

Client init overhead (~24.8s!)

High lazy initialization costs! (> 30s!)

Stabilized latency remains high (380ms)
First Notification

(client) ×5 (cont.)

1st notify

Time (ms)

0 500 1000 1500 2000 2500

0 1 2 3 4 5 3 2 1

Client Resource
First Notification

Container init overhead (~430ms)

Client init overhead (~1.4s!)

Stabilized latency (~1.1s!)
Second Notification

Lazy initialization everywhere

Stabilized request latency still high (170ms)

Stabilized 1st notification (~1.1s)

Resource init overhead (~930ms!)
Internal Profiling: Basics

- **Profiling** data obtained through **sampling**
  (SUN hprof basic profiler on Java 1.4)
  - JVM periodically stopped. Stack of active thread is captured.
  - Result: A set of weighted stack traces. Weight = measures how often the stack was observed.

- **Visualization:**
  Set of weight stacks = **multi-dimensional object**
  - Time (represented by weights)
  - Threads: each trace belongs to a thread
  - Control flow (represented by stacks, reflects use relationships)
  - Code Structure (package organization, class hierarchy, etc.)

- **Projection** (aggregation / collapsing) required

- **Many** possibility. Our choice: **code structure + stack depth**
Methodology III

Sampling yields a set of weighted stack traces (weight reflects time spent)

- Aggregates invocations of the same library.
- Chart w.r.t. position in call stack.
Experimental Set-Up

client

container

Java VM

hprof

profiling data

create

subscribe

add 3

notify 3

destroy

×5

×5

×5
Sharp drop at length 13

Busy waiting related to notification management. Outside request critical path.

Layered structured for upper stack depths

Some very deep traces. Look quite regular beyond depth 28 (recursion?) org.apache.axis predominant
New Experimental Set-Up

+ extra granularity to observe package org.apache.axis

hprof

profiling data
This is a recursion in org.apache.wsdl.symbolTable (web services). Symbol management issue?

Traces of length 13 have disappeared. They were caused by the notification management.

New Results

sun.reflect (reflection)

org.globus.gsi (security)

org.globus.wsrfa
Profiling Breakdown

- Abstracts away **low level** packages (java.*, etc.)

- **Sample breakdown** among “**higher level**” packages:

```
<table>
<thead>
<tr>
<th>Package Name</th>
<th>Samples</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.axis.wsdl</td>
<td>231</td>
<td>21%</td>
</tr>
<tr>
<td>org.apache.axis.encoding</td>
<td>66</td>
<td>6%</td>
</tr>
<tr>
<td>org.apache.axis (others)</td>
<td>113</td>
<td>10%</td>
</tr>
<tr>
<td>org.globus.gsi</td>
<td>249</td>
<td>23%</td>
</tr>
<tr>
<td>org.globus.wsrfs</td>
<td>49</td>
<td>4%</td>
</tr>
<tr>
<td>cryptix.provider.rsa</td>
<td>82</td>
<td>7%</td>
</tr>
<tr>
<td>org.apache.xerces</td>
<td>78</td>
<td>7%</td>
</tr>
<tr>
<td>others</td>
<td>237</td>
<td>21%</td>
</tr>
</tbody>
</table>
```
Profiling Breakdown

- Abstracts away **low level** packages (java.*, etc.)

- **Sample breakdown** among “**higher level**” packages:

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Samples</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.axis.wsdl</td>
<td>231</td>
<td>21%</td>
</tr>
<tr>
<td>org.apache.axis.encoding</td>
<td>66</td>
<td>6%</td>
</tr>
<tr>
<td>org.apache.axis (others)</td>
<td>113</td>
<td>10%</td>
</tr>
<tr>
<td>org.globus.gsi</td>
<td>249</td>
<td>23%</td>
</tr>
<tr>
<td>org.globus.wsrfs</td>
<td>49</td>
<td>4%</td>
</tr>
<tr>
<td>cryptix.provider.rsa</td>
<td>82</td>
<td>7%</td>
</tr>
<tr>
<td>org.apache.xerces</td>
<td>78</td>
<td>7%</td>
</tr>
<tr>
<td>others</td>
<td>237</td>
<td>21%</td>
</tr>
</tbody>
</table>

Symbol management issue?
Profiling Breakdown

- Abstracts away low level packages (java.*, etc.)

- Sample breakdown among “higher level” packages:

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Samples</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.axis.wsdl</td>
<td>231</td>
<td>21%</td>
</tr>
<tr>
<td>org.apache.axis.encoding</td>
<td>66</td>
<td>6%</td>
</tr>
<tr>
<td>org.apache.axis (others)</td>
<td>113</td>
<td>10%</td>
</tr>
<tr>
<td>org.globus.gsi</td>
<td>249</td>
<td>23%</td>
</tr>
<tr>
<td>org.globus.wsrfsuf</td>
<td>49</td>
<td>4%</td>
</tr>
<tr>
<td>cryptix.provider.rsa</td>
<td>82</td>
<td>7%</td>
</tr>
<tr>
<td>org.apache.xerces</td>
<td>78</td>
<td>7%</td>
</tr>
<tr>
<td>others</td>
<td>237</td>
<td>21%</td>
</tr>
</tbody>
</table>

SOAP + XML: 44%
Profiling Breakdown

- Abstracts away **low level** packages (java.*, etc.)

- **Sample breakdown** among “**higher level**” packages:

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Samples</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.axis.wsdl</td>
<td>231</td>
<td>21%</td>
</tr>
<tr>
<td>org.apache.axis.encoding</td>
<td>66</td>
<td>6%</td>
</tr>
<tr>
<td>org.apache.axis (others)</td>
<td>113</td>
<td>10%</td>
</tr>
<tr>
<td>org.globus.gsi</td>
<td>249</td>
<td>23%</td>
</tr>
<tr>
<td>org.globus.wsrdf</td>
<td>49</td>
<td>4%</td>
</tr>
<tr>
<td>cryptix.provider.rsa</td>
<td>82</td>
<td>7%</td>
</tr>
<tr>
<td>org.apache.xerces</td>
<td>78</td>
<td>7%</td>
</tr>
<tr>
<td>others</td>
<td>237</td>
<td>21%</td>
</tr>
</tbody>
</table>

**Security / Cryptography: 30%**
Many Other Visualisation Ways

Time Units

Average Stack Depth

Thread-3, Thread-5, Thread-9, Thread-10, Thread-11, Thread-17, Thread-14, Thread-15, Thread-6, Thread-21, Thread-29, Thread-16, Thread-26, Thread-13, Thread-27, Thread-8, main, Thread-12, Thread-23, Thread-1 Thread, 4, Thread-22, Thread-18, Thread-28, Thread-27, Thread-25, Thread-7, Thread-19, Thread-20, Thread-24, Thread-29, Thread-25, Thread-24, Thread-29, Thread-18, Thread-26, Thread-13, Thread-27, Thread-8, main, Thread-12, Thread-23, Thread-1

Reference Handler, process reaper

(v3.9.2)
Summing Up

- **Globus**
  - **Lazy optimisation**: very high latency on first invocation of operations (up to 30s to set up a resource on a new container!)
  - **Stabilized latencies** still high: ~ 160ms for a round trip request (with authentication turned on)

- No clear culprit. Technology overload: **WSDL, SOAP, security**

- Is lazy optimisation a problem? Yes and No.
Longer Term

- Platforms and technologies come and go
  - Globus is a moving target

- But experimental approaches tend to stay

- And so do development practices
Middleware Practices: Are we doomed?

- Lazy optimization $\rightarrow$ flexibility paradox
- Poor performance $\rightarrow$ abstraction leaking
- Reverse engineering $\rightarrow$ Frankenstein’s return?

Can Cognitive-based Middleware save us?

$\rightarrow$ API are for real beings!
To look further

- **Globus profiling**
  - *The Impact of Web Service Integration on Grid Performance*, François Taïani, Matti Hiltunen, Rick Schlichting, HPDC-14, 2005

- **Large graph reverse engineering**
  - [http://ftaiani.ouvaton.org/7-software/index.html#CosmOpen](http://ftaiani.ouvaton.org/7-software/index.html#CosmOpen)

- **Next Generation Middleware Group at Lancaster**
The End
(Thank you)
Package Activity vs. “Progress”

client
(1 creation, 4 requests, 1 destruction)
Profiling Results (Exclusive, Server)