The Impact of Web Service Integration on Grid Performance

François Taïani
Lancaster University

Matti Hiltunen & Rick Schlichting
AT&T Labs - Research

HPDC-14, The 14th IEEE International Symposium on High Performance Distributed Computing, Research Triangle Park, NC, USA, July 24-27, 2005
Context & Motivation

- New Globus version (3.9.x / 4.0.x): convergence
  - Grid Computing: federating resources (OGSA)
  - Web Services: integrating services (WSRF)

- Web Services and their associated technologies (SOAP, XML, WSDL) are reputed inefficient
  - What is the performance impact on Globus?

- Globus has grown into a large, complex, collaborative middleware (IBM, Apache,...)
  - How to extract meaningful profiling data?

- How to profile a complex piece of software?
- What does it tell us about Globus?
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)
Outline

- Introduction: Motivation and Approach
- Black Box Profiling: Set-Up and Results
- Sample Based Profiling: Approach and Results
- Conclusion
Outline

- Introduction: Motivation and Approach
- Black Box Profiling: Set-Up and Results
- Sample Based Profiling: Approach and Results
- Conclusion
Black-Box Profiling: Approach

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

---

![Diagram](image-url)

**client**

- influence of resource init
- influence of client init
- influence of container init

**container**

- create
- subscribe
- add 3
- notify 3
- destroy

- averaging

× 5

× 5

× 5

× 5

× 50 (10 000 invocations)
Resource Set-Up

- Client
- Create
- Subscribe
- Cont.

Bar chart showing time (ms) against client process and resource, with an indication of a factor of 5 increase.
Resource Set-Up

Container init overhead (~8.2s!)

Client init overhead (~24.8s!)
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) 1<sup>st</sup> notify (cont. ×5)
First Notification

- Container init overhead (~430ms)
- Client init overhead (~1.4s!)
- Stabilized latency (~1.1s!)
Second Notification

Stabilized latency 1\textsuperscript{st} notification (~1.1s)

Resource init overhead (~930ms!)
Second Notification

Lazy initialization everywhere

Stabilized request latency still high (170ms)

Resource init overhead (~930ms!)

Stabilized latency after 1st notification (~1.1s)
Outline

- Introduction: Motivation and Approach
- Black Box Profiling: Set-Up and Results
- Sample Based Profiling: Approach and Results
- Conclusion
Sample Based Profiling: Introduction

- **Goal:** *relate* observed latencies to Globus internal structure

- **Profiling** data obtained through **sampling** (SUN hprof basic profiler)
  - 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.)
Program visualization

- Problem studied for quite a long time now.
- **Projection** (aggregation / collapsing) required
- *Many* possibility.
  - Our goal: related profiling to software structure
  - Our choice: **package aggregation + stack depth**

Tracing calls reveals the software structure.
Sample Based Profiling: Example

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

Operations:
- Create
- Subscribe
- Add 3
- Notify 3
- Destroy

Profiling data
Container Profiling: Results
Container Profiling: Results

- Sharp drop at length 13
- Layered structured for upper stack depths
- Some very deep traces. Look quite regular beyond depth 28 (recursion?)
- org.apache.axis predominant
Container Profiling: Results

- Sharp drop at length 13
- Layered structured for upper stack depths
- Busy waiting related to notification management. Outside request critical path.
- 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

profiling data
New Results

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

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

org.globus.wsrf

org.globus.gsi (security)

sun.reflect (reflection)
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.wsrft</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.wsrfr</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.wSRF</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%**
(Temporary) Conclusion on Globus

- **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. A mix of factors: WSDL, SOAP, security

- Is lazy optimisation a problem? Yes and No.
  - Brand new version. 3.9.4 numbers already better than 3.9.2!
  - Containers not supposed to be started frequently
  - Globus services are there to manage very long running jobs. A few seconds does not really matter.
  - But points at some applications for which Globus (in its present form) would be clearly ill chosen
Use of simple and well known profiling techniques

Visualisation was adapted to scale up to the complexity of a software like Globus

The diagrams we used don’t contain all the answers:
- They can be best seen as maps to guide further steps
- Different kinds of projection actually useful

Interesting complexity related problems:
- Which is the best “semantically relevant” level to project profiling traces? Too low: no meaning. Too high: no details.
- Can we leverage the “middleware” nature of Globus to obtain finer profiling data with the same lightweight tools?
The End
(Thank you)