The Impact of Web Service Integration on Grid Performance

François Taïani
Lancaster University

Matti Hiltunen & Rick Schlichting
AT&T Labs - Research

IBM Hursley Technical Exchange (HTX'06), May 23 2006
Preamble: 2 Years Ago

- New Globus version (3.9.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: **large, complex, collaborative** middleware
  - How to **extract meaningful** profiling data?

- How to **profile a complex** piece of software?
- Both a **profiling and a reverse engineering** problem
Our First Experiment

- 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?

[Globus 3.9.2, Java 1.4, no security]
Program visualization: a few Notions

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

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

- Tracing calls reveals the software structure.
Methodology I

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

lib 1
lib 2
lib 3
Package Activity vs. Stack Depth

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

[Globus 3.9.2, Java 1.4, no security]
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!
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)
Package Activity vs. Stack Depth

(client, percentage view)

F. Taïani. IBM HTX'06
Package Activity vs. Stack Depth

(client, percentage view)

Calls vs. Stack Depth

F. Taïani. IBM HTX'06
Package Activity vs. Stack Depth

(client, percentage view)

Very long stack $\rightarrow$ probably recursive parsing!

XML used by org.apache.axis, not by Globus!

F. Taïani. IBM HTX'06
What does it tell us?

- Most **local invocations** (89%) related to **XML** (org.apache.xerces, org.apache.xml).
- The parsing due to **Axis** (SOAP/Java bridge)
- **Very long stacks** (up to 57 frames!): recursive parsing loop?
- **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 / good is the platform really?
- Can we do the same kind of “structural” projection of profiling data?
- If yes, is it useful?
Our Line of Attack

- 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
  - With default security (X509, ssl)

- Experimental set-up:
  - **Standalone SMP** server running 4 Intel Xeon @ 1.6GHz
  - **No network cost** involved!
  - **Avoids context switching** overhead!
  - Globus **4.0.1** used (latest GT4 release 4.0.2)
Outline

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

- Preamble: 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 showing client-container interactions with various operations and waits, indicating influences of resource init, container init, and client init with factors of 5 and 50 averaged over 10,000 invocations.]
Resource Creation

- Client
- Subscribe
- Create
- Cont. x5

Time (ms)

Client Process

Resource

F. Taïani. IBM HTX'06
Resource Creation

Container init overhead (~8.2s!)

Client init overhead (~24.8s!)
Resource Creation

High lazy initialization costs! (> 30s!)

Stabilized latency remains high (380ms)
First Notification

![Diagram showing time in milliseconds for different clients and resources, with notation of a 5x increase in time for the first notification.]
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)
Second Notification

Stabilized latency 1st notification (~1.1s)

Resource init overhead (~930ms!)
Second Notification

Lazy initialization everywhere

Stabilized request latency still high (~170ms)

Resource init overhead (~930ms!)

F. Taïani. IBM HTX'06
Outline

- Preamble: 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 threads 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**

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

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

Sampling yields a set of \textit{weighted} stack traces (weight reflects time spent)

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

\textbf{EXCLUSIVE}

\textbf{INCLUSIVE}
Experimental Set-Up

client

create
subscribe
add 3
notify 3
destroy

×5

container

Java VM

hprof

profiling data
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
- 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
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?

sun.reflect (reflection)

org.globus.gsi (security)

org.globus.wsrfs
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.wsrfl</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>

100%
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.wsdI</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.wsr</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 encryption and authentication turned on)

- No clear culprit. A mix of factors: WSDL, SOAP, security

- **Is lazy optimisation a problem? Yes and No.**
  - Measurements improve with versions (4.0.1 > 3.9.4 > 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
Profiling Complexity

- 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?
Looking Windward

- WS-* standard stack far from stable yet
  - March 2006: roadmap to replace WS-RF (among others) by major players (MS, HP, IBM)
  - Expected impact on SOAP-RPC limited (hopefully)

- More general debate: is WS-* good for the grid?
  - RESTafarian controversy
  - Heavyweight WS-* vs. lightweight XML

Big middleware houses (IBM, HP, MS)
Buzzword: SOA

Big service providers (Google, Amazon, FedEx)
Buzzword: Web 2.0
Reference


[available at http://ftaiani.ouvaton.org]
The End
(Thank you)