WWV 2013

Deconstructing Complex Distributed Platforms

A Report from the Trenches

François Taïani
Joint work with

- Matti Hiltunen
- Rick Schlichting
- Shen Lin
- Tom Ormerod
- Linden Ball
Distributed Software Today

- External services
- Standards
- External developers
- Large, complex, multi-party
- Geosocial app, est. 2009
- 10M Users
- 3
Why deconstruct software?  
(in particular distributed)

- Needed for **diagnosis / modifications**
  - Harden software (FT, replication)
  - To address non-functional issues (e.g. performance)

- Assembled from **third party** components
  - Unfamiliar software
  - Frankenstein effect
Why is it difficult?

- Software easier to construct than to understand

- Particularly true for emergent dynamic behaviours
  - e.g. dependability and performance
  - arise from interactions within/without target system

- Dynamic observation data difficult to make sense of
  - large size (up to 1,000,000 invocations for a few sec)
  - likely to involves poorly understood parts
Example 1: Replication

- Replicating COTS applications: **non-determinism**
  - Observe mutex activities
  - Link mutexes to **request life cycle**

**Approach: dynamic reverse engineering**

- **Application**
- **Middleware services** (RPC, pub/sub, ...)
- **Syscalls, syslibs** (synchronisation, memory management...)
- **OS**
Dynamic Reverse Engineering

ORBacus
1 Request

one request,
2065 individual invocations,
over 50 C-functions and 140 C++ classes.

How to analyse this?
Example 2: Profiling

- **Globus** (Argonne): ref. implementation for Grid
  - Grid Computing + Web Services

- Transition to WS stack (Version 3.9.x, 2005)
  - within a short time (a few months)
  - large, complex, collaborative (IBM, Apache,...) ➔ reuse

- **But … poor performances**
  - Up to 30s to create a simple distributed object (counter)
  - Up to 2s for a roundtrip remote add operation
Profiling

■ Large piece of software (3.9.x)
  ➔ 123,839 lines in Java (w/oreused libraries)

■ Many libraries layered
  ➔ XML, WSDL (Descr. Lang), WSRF (Resource Fwork)
  ➔ Axis (SOAP), Xerces (XML Parsing), com.ibm.wsdl

■ Java: exhaustive tracing (outside the JVM libs)
  ➔ client : 1,544,734 local method call (sic)
  ➔ server : 6,466,652 local method calls (sic) [+time out]
Profiling

How to analyse this?

The Impact of Web Service Integration on Grid Performance. Taïani, Hiltunen, Schlichting, HPDC-14, 2005
The Deconstruction Problem

- Commonalities
  - Large scale, software intensive, composite
  - Unfamiliar

- Our general philosophy
  - Pragmatic, semi-automatic, interactive
Outline

■ Motivation

■ Example 1:
  ➔ Analysing locking behaviour in industrial ORB

■ Example 2:
  ➔ Visualizing performance anomalies in Grid MW

■ Some concluding words
Back to Example 1

- Original goal
  - Control non-determinism in industry grade middleware
  - Observation of mutex activities
  - Link to request life cycle
Back to Example 1

- Original goal
  - Control non-determinism in industry grade middleware
  - Observation of mutex activities
  - Link to request life cycle

- Complex & layered system

- Hard to observe and analyse
  - Problem 1: Observation cost
  - Problem 2: Cross-layer entangling
Problem 1: Observation costs

- Time consuming
- Different behaviour!
Problem 2: Cross-layer entangling

Can we reconstruct main’s behaviour?

OS

dummy1
dummy2
Problem 2: Cross-layer entangling

Much more complex than original code

Program behaviour not apparent

Reason: mixes several abstraction levels
Approach: “interferometry”

■ **Limit** observation costs: *more for less*
  - capture stack traces (more)
  - limit observation points to component boundaries (less)

■ **Filter** out unneeded data: “interferometry”
  - graph manipulation script language
  - reusable filters

‘cheap’ but post-processing needed

no turbulence but very costly

F. Taiani
Tackling Observation Costs

```
#0  0x405c8f20 in send () from /lib/libpthread.so.0
#1  0x4015804a in omni::tcpConnection::Send ()
#2  0x4011364c in omni::giopStream::sendChunk ()
[...]
#8  0x400bdb0a in omniAsyncWorker::run ()
#9  0x405abbdf in omni_thread_wrapper ()
#10 0x405c2bf0 in pthread_start_thread ()
#11 0x405c2c6f in pthread_start_thread_event ()
```
Result from Observation

ORBacus
1 Request

one request,
2065 individual invocations,
over 50 C-functions and 140 C++ classes.

Cross layer entangling
Tackling Entangling

A number of graph operators

Interactive graph scripting language

- selection
  put ::pthread_create* G CREATE

- recursive extension
  forward CREATE G

- boolean algebra
  remove ::pthread_create* CREATE

- temporal operations
  fuse ::pthread_create* ::pthread_start_thread* G
Disentangling Script

Pattern-based, reusable

2065 individual invocations, over 50 C-functions and 140 C++ classes.

// [remove pthread]
put ::recv"*" GlobalGraph R
put ::send"*" GlobalGraph R
put Hello_impl::say_hello"*" GlobalGraph R
backward R GlobalGraph
put ::lsf_thread_adapter"*" ABS
put JTC* ABS
put ::__libc_start_main't1-0 ABS
put ::run't1-9 ABS
put OB::ORBControl::initializeRootPOA't1-11 ABS
put OBPortableServer::POAManagerFactory_impl::create_poa_manager't1-12 POA forwN 3 POA R
add POA ABS
put OBPortableServer::POAPolicies::POAPolicies't1-42 ABS
put OB::DispatchStrategyFactory_impl::* ABS
abstract ABS R
abstract ABS GlobalGraph
remove ::accept't1-882 R
absPatern ::recv't18-828 R
absPatern ::recv't18-1309 R
absPatern ::recv't18-1908 R
put OB::Upcall::Upcall't1-1083 U
backward U GlobalGraph
add U R
put OB::ThreadPool::add't4-1224 TP
put OB::ThreadPool::get't4-1265 TP
backward TP GlobalGraph
add TP R
put OB::GIOPServerStarterThreaded::StarterThread::* ABS2
put OB::DispatchThreadPool_impl::* ABS2
put OB::DispatchRequest_impl::* ABS2
put OB::POAOAInterface_impl::* ABS2
abstract ABS2 R
absPatern OB::GIOPServerWorker::executeRequest't8-* R

Result
Use

- Modelling
- Analysis

\[ G_1 = \{R4; R5\} \]

\[ G_2 = \{R2; R3\} \]

\[ G_3 = \{R1\} \]

requestBeforeApplication

RequestContentionPoint

injectRequestAtCommunicationLevel

skipCallToApplication

requestAfterApplication
Deterministic replication

- link OS level ↔ app level
- use **reflective** approach
- instrument MW to control non-determinism

Example 1:
- Analysing locking behaviour in industrial ORB

Example 2:
- Visualizing performance anomalies in Grid MW

Some concluding words
Example 2: Profiling

- 30s for first RPC
- 1,544,734 method calls
- Why?

Our take: sampling + visualisation
Sample-based profiling

client

create
subscribe
add 3
notify 3
destroy

×5

container

Java VM

hprof

snapshots

regular interval

profiling data
Sample-based profiling

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

lib3.Blood .flow
lib3.Pressure.foo

lib2.Muscle.stop
lib2.Nerve .transmit
lib3.Signal.travel
Sample-based profiling

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

- Problem: **Data explosion remains.** On Globus:
  - 55550 method invocations
  - 1861 methods
  - 724 classes
  - 182 Java packages.
  - 32 threads
Visualising Dynamic Data

- Strategies: extract **salient features**
  - collapse recurring patterns (e.g. Jinsight)
  - remove and focus (e.g. CosmOpen)
  - data-mining (e.g. PCA in Xu et. al, SOSP’09)

- Our strategy
  - use **structural information** in dynamic data
    e.g. `org.apache.axis.utils.ClassUtils.forName()`
  - vary ‘**local abstraction**’ level at which data is shown

- Contributions
  - application of structural compaction to **dynamic data**
  - algorithm for **localised structural compaction**
  - prototype and **explorative user study**
Compaction

- Only highest level packages visible
Progressive exploration

- Different levels of compaction in different parts of graph
  ➞ including for the same package (here lib3)
Towards anomaly comprehension: Using structural compression to navigate profiling call-trees S. Lin, F. Taiani, T. C. Ormerod, L. J. Ball, ACM SoftVis'2010
http://ftaiani.ouvaton.org/7-software/profvis.html
Evaluation: User Study

- Visual presentation high impact on exploration pattern
  ➔ users go deep w/ TreeTable, tend to hover w/ ProfVis
Evaluation: User Study

- Understanding
  - slight advantage for Provis

- But
  - very small study!
  - informally, users preferred TreeTable

- Questions raised
  - acceptability
  - ergonomics
  - design combination
Conclusion

■ Distributed Systems
  ➡ Increased reused → layered systems
  ➡ Frankenstein effect: abstraction leaks

■ Pragmatic approaches to dynamic reverse engineering
  ➡ addresses complex layered software
  ➡ aim: help user construct mental models of system

■ Future questions
  ➡ To which extent can this be automated (slicing? SA?)
  ➡ Useful for verification?
Thank you
(Some) references

- Jerding DF, Stasko JT, and Ball T. Visualizing interactions in program executions. Proc. of the 19th Int. Conf. on Software Engineering (ICSE’97), ACM Press, 1997; 360-370
(Some) references (cont)


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?
Reconstructing a call tree

- Problem: stack traces are ambiguous
- Choice: “smallest” compatible call tree, a variant of smallest prefix tree (with timestamps) can be characterised formally.

2 identical traces
Revisiting pthread

fuse ::pthread_create* ::pthread_start_thread*
Revisiting pthread

'Reconstructing' data-flow from temporal succession

fuse ::pthread_create* ::pthread_start_thread* G
Revisiting pthread

fuse ::pthread_create* ::pthread_start_thread* G
put ::pthread_create* G CREATE
forwN 1 CREATE G
remove ::pthread_create* CREATE
remove ::pthread_start_thread* CREATE
forward CREATE G
exclude CREATE G
absPatern ::pthread_create* G
absPatern ::pthread_start_thread* G

Generic script: applies to any pthread code
Case study

- 3 C/C++ industry-grade CORBA products
  - ORBacus, omniORB, TAO

- Set up
  - ~ 60 breakpoints (locks, memory, threading, callbacks)
  - one ping-pong request
  - 1GHz Pentium III, Linux kernel 2.4, gdb 5.1-1

- Observation **overheads** (Orbacus)
  - non-instrumented: < 1s
  - fully instrumented: 1h 2m 11s
  - lock tracing disabled during initialisation: 4m 53s
## Stack capture speedup

- Breakpoint activation = cost
- Reconstructed tree = added value
- Ratio = speedup

(with lock tracing disabled during initialisation)

<table>
<thead>
<tr>
<th>ORB</th>
<th>threads</th>
<th>traces</th>
<th>frames</th>
<th>invocations</th>
<th>invocations/traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORBACUS 4.1</td>
<td>8</td>
<td>658</td>
<td>9178</td>
<td>2066</td>
<td>3.13</td>
</tr>
<tr>
<td>omniORB 4</td>
<td>7</td>
<td>1828</td>
<td>16807</td>
<td>3088</td>
<td>1.68</td>
</tr>
<tr>
<td>TAO 1.2.1</td>
<td>6</td>
<td>512</td>
<td>11260</td>
<td>1352</td>
<td>2.64</td>
</tr>
</tbody>
</table>

```
#0 0x405c8f20 in send () from /lib/libpthread.so.0  
#1 0x4015804a in omni::tcpConnection::Send ()  
#2 0x4011364c in omni::giopStream::sendChunk ()  
[..]  
#8 0x400bdb0a in omniAsyncWorker::run ()  
#9 0x405abbdf in omni_thread_wrapper ()  
#10 0x405c2bf0 in pthread_start_thread ()  
#11 0x405c2c6f in pthread_start_thread_event ()
```