High-performance Distributed Object Computing with Real-time CORBA

Douglas C. Schmidt
Washington University, St. Louis

http://www.cs.wustl.edu/~schmidt/schmidt@cs.wustl.edu

Motivation

- Typical state of affairs today is the “Distribution Crisis”
  - Computers and networks get faster and cheaper
  - Communication software gets slower, buggier, more expensive

- Accidental complexity is one source of problems, e.g.,
  - Incompatible software infrastructures
  - Continuous rediscovery and reinvention of core concepts and components

- Inherent complexity is another source of problems
  - e.g., latency, partial failures, partitioning, causal ordering, etc.

Candidate Solution: CORBA

Goals

1. Simplify development of distributed applications
2. Provide flexible foundation for higher-level services

OMA Reference Model Interface Categories
Example 1: Distributed Medical Imaging

Distributed Objects in Medical Imaging

• "Blob" = Binary Large Object

Example 2: Real-time Avionics

Observations

• CORBA is well-suited for certain communication requirements and certain network environments
  - e.g., request/response or one-way messaging over low-speed Ethernet or Token Ring

• However, current CORBA implementations exhibit high overhead for other types of requirements and environments
  - e.g., bandwidth-intensive and delay-sensitive applications over high-speed networks

• Performance limitations will ultimately impede adoption of CORBA
Key Research Questions

- "Can CORBA be used for performance-sensitive applications on high-speed networks?"
  - Goal is to determine this empirically

- "What are the strategic optimizations for Gigabit CORBA?"
  - Goal is to maintain strict CORBA compliance

- "What changes are required to provide Real-time CORBA?"
  - Goal is to provide end-to-end QoS guarantees

Pinpointing CORBA Overhead

- **Presentation layer overhead**
  - e.g., typed and untyped data

- **Data manipulation and data copying overhead**
  - e.g., message management

- **Demultiplexing and operation dispatching overhead**
  - e.g., layered and de-layered demultiplexing

- **OS/network/protocol integration**
  - e.g., ATM/host adapters, resource reservation and scheduling

General Path of CORBA Requests

Experimental Setup

- **Enhanced version of TTCP**
  - TTCP measures end-to-end data/request transfer
  - Enhanced version compares C, ACE C++ wrappers, Orbix 2.0.1 and VisiBroker 2.0, and Blob Streaming

- **Parameters varied**
  - 100 Mbytes of typed data
    - Types included scalars, floats, structs, and sequences
  - Sender buffer sizes ranged from 1K to 128K
  - Socket queues were 8k (default) and 64k (maximum)
  - Network was 155 Mbps ATM and “loopback”
TTCP Configuration for Blob Streaming (Push Model)

Sender
1: send(buf)
2: connect
3: write(buf)
4: forward
5: read(buf)

Blob Store
6: send(buf)
7: ack

Blob_Xport Stub

Src Blob Proxy
Dest Blob Proxy

Skel

Push Model Performance over ATM and Ethernet

C, ACE, Orbis, and Blob Streaming over ATM and Ethernet

Mbits/sec

0 10 20 30 40 50 60 70

Blob chunk size in megabytes

C/64k window
ACE/64k window
Blob Streaming/64k window
Orbis Sequence/64k window
Orbis String/64k window
C/8k window
ACE/8k window
Blob Streaming/8k window
Orbis Sequence/8k window
Orbis String/8k window
All Ethernet results

TTCP Configuration for Blob Streaming (Pull Model)

Receiver
1: receive(buf)
2: connect
3: send(buf)
4: write(buf)
5: forward
6: read(buf)
7: ack

Blob Store

Blob_Xport Skel

Src Blob Proxy
Dest Blob Proxy

Skel

Pull Model Performance over ATM

Mbits/sec

0 5 10 15 20 25 30 35 40 45 50 55 60

Blob chunk size in megabytes

Push and Pull Models of Blob Streaming over ATM

Push Model/64k window
Push Model/8k window
Pull Model/64k window
Pull Model/8k window
All Ethernet results
### High-Cost Functions

**C and ACE C++ Tests**

- Transferring 100 Mbytes with 1 Mbyte buffers

<table>
<thead>
<tr>
<th>Test</th>
<th>%Time</th>
<th>#Calls</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C sockets (sender)</td>
<td>93.9</td>
<td>112</td>
<td>write</td>
</tr>
<tr>
<td>C sockets (receiver)</td>
<td>93.2</td>
<td>13,085</td>
<td>read</td>
</tr>
<tr>
<td>ACE C++ wrapper (sender)</td>
<td>94.4</td>
<td>112</td>
<td>write</td>
</tr>
<tr>
<td>ACE C++ wrapper (receiver)</td>
<td>3.2</td>
<td>110</td>
<td>read</td>
</tr>
</tbody>
</table>

**Orbix String and Sequence**

<table>
<thead>
<tr>
<th>Test</th>
<th>%Time</th>
<th>#Calls</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orbix Sequence (sender)</td>
<td>53.5</td>
<td>127</td>
<td>write</td>
</tr>
<tr>
<td></td>
<td>35.1</td>
<td>223</td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>7.3</td>
<td>1,108</td>
<td>memcpy</td>
</tr>
<tr>
<td>Orbix Sequence (receiver)</td>
<td>84.6</td>
<td>12,848</td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>12.4</td>
<td>1,064</td>
<td>memcpy</td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>101</td>
<td>write</td>
</tr>
<tr>
<td>Orbix String (sender)</td>
<td>45.0</td>
<td>127</td>
<td>write</td>
</tr>
<tr>
<td></td>
<td>35.1</td>
<td>223</td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>10.8</td>
<td>1,315</td>
<td>strlen</td>
</tr>
<tr>
<td></td>
<td>6.0</td>
<td>1,108</td>
<td>memcpy</td>
</tr>
<tr>
<td>Orbix String (receiver)</td>
<td>70.7</td>
<td>12,443</td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>16.1</td>
<td>2,142</td>
<td>strlen</td>
</tr>
<tr>
<td></td>
<td>10.0</td>
<td>1,064</td>
<td>memcpy</td>
</tr>
<tr>
<td></td>
<td>3.0</td>
<td>101</td>
<td>write</td>
</tr>
</tbody>
</table>

**Blob Streaming**

<table>
<thead>
<tr>
<th>Test</th>
<th>%Time</th>
<th>#Calls</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlobStreaming (sender)</td>
<td>48.8</td>
<td>327</td>
<td>write</td>
</tr>
<tr>
<td></td>
<td>44.8</td>
<td>232</td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>2,055</td>
<td>memcpy</td>
</tr>
<tr>
<td>BlobStreaming (receiver)</td>
<td>77.2</td>
<td>12,846</td>
<td>read</td>
</tr>
<tr>
<td></td>
<td>16.4</td>
<td>12,734</td>
<td>memcpy</td>
</tr>
<tr>
<td></td>
<td>1.4</td>
<td>102</td>
<td>write</td>
</tr>
</tbody>
</table>

### C/C++ Remote Data Transfer Results

A graph showing throughput in Mbps vs. sender buffer size in KBytes.
Orbix Remote Data Transfer

Results

Throughput in Mbps

Sender Buffer size in KBytes

Analysis of Orbix Overhead

Analysis of VisiBroker Overhead
Summary of Throughput Results

- For bytestreams
  - VisiBroker performed > 80% cf C/C++ versions
  - Orbix performed 70% cf C/C++ versions

- For scalars and floats
  - Remote: 75–80% cf C/C++ versions
  - Loopback: Orbix performed 65–68% cf C/C++ versions, VisiBroker achieved similar throughput for large sender buffer sizes

- For structures
  - Remote: 31% cf C/C++ versions
  - Loopback: 10% cf C/C++ versions
Multi-Protocol Support

I/O Subsystem Optimizations

Real-time Scheduling Optimizations

Concluding Remarks

- CORBA is a promising architecture for distributed computing

- Conventional CORBA implementations are not tuned for high-performance or real-time systems
  - Note, low-speed networks often hide performance overhead

- Ultimately, an integrated approach is the best solution

- Optimizations must be applied at multiple layers
  - e.g., network/OS/protocol/ORB