Measuring the Performance of CORBA for High-speed Networking

Douglas C. Schmidt
schmidt@cs.wustl.edu

http://www.cs.wustl.edu/~schmidt/
Washington University, St. Louis

Introduction

- Distributed object computing (DOC) frameworks are 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 DOC implementations exhibit high overhead for other types of requirements and environments
  - e.g., bandwidth-intensive and delay-sensitive streaming applications over high-speed ATM or FDDI

Outline of Talk

- Outline communication requirements of distributed medical imaging domain
- Compare performance of several network programming mechanisms:
  - Sockets
  - ACE C++ wrappers
  - Two CORBA implementations (ORBeline and Orbix)
- Discuss how to use distributed object computing frameworks efficiently and effectively

Distributed Medical Imaging
Distributed Objects in Medical Imaging Systems

- Image servers have the following responsibilities and requirements:
  - Efficiently store/retrieve large medical images
  - Respond to queries from Image Locator Servers
  - Manage short-term and long-term image persistence

Image Server System Architecture

Motivation for CORBA

- Simplifies application interworking
  - CORBA provides higher level integration than traditional “untyped TCP byte streams”

- Provides a foundation for higher-level distributed object collaboration
  - e.g., Windows OLE and the OMG Common Object Service Specification (COSS)

- Benefits for distributed programming similar to OO languages for non-distributed programming
  - e.g., encapsulation, interface inheritance, and object-based exception handling

CORBA Overview

- CORBA specifies the following functions of an Object Request Broker (ORB)
  - Interface Definition Language (CORBA IDL)
  - A mapping from CORBA IDL onto C, C++, and Smalltalk
  - An Interface Repository
    - Contains meta-info that can be queried at runtime
    - A Dynamic Invocation Interface
      - Used to compose method requests at run-time
    - A Basic Object Adaptor (BOA)
      - Allows developers to integrate their objects with an ORB
CORBA Services

- CORBA provides the following mechanisms
  - Parameter marshalling
  - Object location
  - Object activation
  - Replication and fault tolerance

- COSS extends CORBA to provide services like
  - Event services
  - Naming services
  - Transactions
  - Object lifecycle management

Key Research Question

Can CORBA be used to transfer medical images efficiently over high-speed networks?

- Our goal was to determine this empirically before adopting the CORBA communication model wholesale

Performance Experiments

- Enhanced version of TTCP
  - TTCP measures end-to-end, one-way bulk data transfer
  - Enhanced version tests C, ACE C++, wrappers, and CORBA

- Parameters varied
  - 64 Mbytes of data buffers ranging from 1 Kbyte to 128 Kbyte (by powers of 2)
  - Socket queues were 8k (default) and 64k (maximum)
  - Networks were 155 Mbps ATM and 10 Mbps Ethernet

- Compiler was SunC++ 4.0.1 using highest optimization level

Network/Host Environment

BAY NETWORKS LATTISCELL ATM SWITCH (16 PORT, OC3 155MBPS/PORT, 9,180 MTU)
Sparcstation 20 model 50s (ENI ATM ADAPTORS AND ETHERNET)
TTCP Configuration for C and ACE C++ Wrappers

1: write(buf)
2: forward
3: read(buf)

ATM SWITCH

TTCP Configuration for CORBA Implementations

1: send(buf)
2: forward
3: send(buf)

CORBA Implementations

- 2 implementations of TTCP using 2 versions of CORBA
  - IDL string and IDL sequence
    ```
    typedef sequence<char> ttcp_sequence;
    
    interface TTCP_Sequence
    {
      oneway void send (in ttcp_sequence ttcp_seq);
    };
    
    interface TTCP_String
    {
      oneway void send (in string ttcp_string);
    };
    ```
  - Orbix 1.3 and ORBeline 1.2
    - Couldn’t directly reuse source code since neither ORB supported same IDL — C++ mapping
    - Also, neither ORB supported CORBA 2.0

CORBA Sender Implementation

- Obtain reference to target objects via `bind` factory:

  ```
  // Use locator service to acquire bindings.
  TTCP_String *t_str = TTCP_String::bind () ;
  TTCP_Sequence *t_seq = TTCP_Sequence::bind () ;
  ```
  ```
  // ...
  // String transfer.
  char *buffer = new char[buffer_size];
  // Initialize data in char * buffer...
  while (--buffers_sent > 0)
    t_str->send (buffer);
  ```
  ```
  // Sequence transfer.
  ttcp_sequence sequence_buffer;
  // Initialize data in TTCP_Sequence buffer...
  while (--buffers_sent > 0)
    t_seq->send (sequence_buffer);
  ```
CORBA Receiver Implementation

- Implementation class for IDL interface that inherits from automatically-generated CORBA skeleton class

```cpp
class TTCP.Sequence_i :
  virtual public TTCP.SequenceBOAImpl
{
  public:
    TTCP.Sequence_i (void); nbytes_ (0) {} 
    // Upcall invoked by the CORBA skeleton.
    virtual void send (const ttcp_sequence &ttcp_seq,
                       CORBA::Environment &IT_env)
    {
      this->nbytes_ += ttcp_seq.length; 
    } 
  } 
private:
  // Keep track of bytes received.
  u_long nbytes_; 
};
```

CORBA Receiver Main

- Initializes object implementations and goes into CORBA event loop

```cpp
int main (int argc, char *argv[])
{
  // Implements the Sequence object.
  TTCP.Sequence_i ttcp_sequence;
  // Implements the String object.
  TTCP.String_i ttcp_string;
  // Tell the ORB that the objects are active.
  CORBA::BOA::impl_is_ready ();
  // NOTREACHED */
  return 0;
}
```

Performance over ATM and Ethernet

![Performance over ATM and Ethernet](chart1.png)

C and ACE Performance over ATM and Ethernet

![C and ACE Performance over ATM and Ethernet](chart2.png)
Orbix and ORBeline Performance over ATM and Ethernet

Primary Sources of Overhead for CORBA

- Data copying
- Demultiplexing
- Memory allocation
- Presentation layer formatting

High-Cost Functions

- C and ACE C++ Tests
  - Transferring 64 Mbytes with 128 Kbyte buffers

<table>
<thead>
<tr>
<th>Test</th>
<th>%Time</th>
<th>#Calls</th>
<th>msec/call</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C sockets (sender)</td>
<td>99.6</td>
<td>527</td>
<td>92.8</td>
<td>_write</td>
</tr>
<tr>
<td>C sockets (receiver)</td>
<td>99.3</td>
<td>7201</td>
<td>6.2</td>
<td>_read</td>
</tr>
<tr>
<td>ACE C++ wrapper (sender)</td>
<td>99.4</td>
<td>527</td>
<td>87.3</td>
<td>_write</td>
</tr>
<tr>
<td>ACE C++ wrapper (receiver)</td>
<td>99.6</td>
<td>7192</td>
<td>6.2</td>
<td>_read</td>
</tr>
</tbody>
</table>

High-Cost Functions (cont’d)

- Orbix String and Sequence Tests

<table>
<thead>
<tr>
<th>Test</th>
<th>%Time</th>
<th>#Calls</th>
<th>msec/call</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orbix Sequence (sender)</td>
<td>94.6</td>
<td>532</td>
<td>88.1</td>
<td>_write</td>
</tr>
<tr>
<td>Orbix Sequence (receiver)</td>
<td>92.7</td>
<td>7860</td>
<td>6.1</td>
<td>_read</td>
</tr>
<tr>
<td>Orbix String (sender)</td>
<td>89.0</td>
<td>532</td>
<td>86.6</td>
<td>_write</td>
</tr>
<tr>
<td>Orbix String (receiver)</td>
<td>86.3</td>
<td>7744</td>
<td>5.7</td>
<td>_read</td>
</tr>
<tr>
<td>ACE String (sender)</td>
<td>86.3</td>
<td>7744</td>
<td>5.7</td>
<td>strlen</td>
</tr>
<tr>
<td>ACE String (receiver)</td>
<td>5.5</td>
<td>6740</td>
<td>0.4</td>
<td>strlen</td>
</tr>
<tr>
<td>ACE C++ wrapper (sender)</td>
<td>4.5</td>
<td>2581</td>
<td>0.9</td>
<td>memcp</td>
</tr>
<tr>
<td>ACE C++ wrapper (receiver)</td>
<td>4.5</td>
<td>2581</td>
<td>0.9</td>
<td>memcp</td>
</tr>
</tbody>
</table>
High-Cost Functions (cont’d)

- ORBeline String and Sequence Tests

<table>
<thead>
<tr>
<th>Test</th>
<th>Time</th>
<th>#Calls</th>
<th>msec/call</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORBeline Sequence 89.0</td>
<td>5.1</td>
<td>7222</td>
<td>0.3</td>
<td>memcpy</td>
</tr>
<tr>
<td>(receiver)</td>
<td>3.3</td>
<td>1071</td>
<td>1.3</td>
<td>poll</td>
</tr>
<tr>
<td>ORBeline String 8.3</td>
<td>5.4</td>
<td>920</td>
<td>3.2</td>
<td>strcpy</td>
</tr>
<tr>
<td>(sender)</td>
<td>4.3</td>
<td>5901</td>
<td>0.4</td>
<td>memcpy</td>
</tr>
<tr>
<td></td>
<td>3.9</td>
<td>1728</td>
<td>1.2</td>
<td>strlen</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>1032</td>
<td>0.6</td>
<td>_sigaction</td>
</tr>
</tbody>
</table>

Evaluation and Recommendations

- Understand communication requirements and network/host environments
- Measure performance empirically before adopting a communication model
  - Low-speed networks often hide performance overhead
- Insist CORBA implementors provide hooks to manipulate options
  - e.g., setting socket queue size with ORBeline was hard
- Increase size of socket queues to largest value supported by OS
- Tune the size of the transmitted data buffers to match MTU of the network

Evaluation and Recommendations (cont’d)

- Use IDL sequences rather than IDL strings to avoid unnecessary data access and copying
- Use write/read rather than send/recv on SVR4 platforms
- Long-term solution:
  - Optimize DOC frameworks
  - Add streaming support to CORBA specification
- Near-term solution for CORBA overhead on high-speed networks:
  - Integrate DOC frameworks with OO encapsulation of network programming interfaces

Concluding Remarks

- To be effective for use with performance-critical applications over high-speed networks, CORBA implementations must be optimized
- Key optimization points are illustrated above