PERFORMANCE OF JAVA DISTRIBUTED OBJECT MODELS

Matt Young
November 15, 2011
Overview

- Distributed Systems
- Java
- Communication
- Serialization
- Java Communication Models
- Studies/Results
- Summary
What is a Distributed System?

- A collection of independently operating computers that function as a single system
- Users are unaware the system is distributed
Java

- Very popular in distributed systems:
  - Code is executed in a Java Virtual Machine
  - Can easily be ported to different environments
Communication

- One of the most critical requirements for a distributed system is communication.

- Communication can be:
  - Between clients (users) and remote server
  - Between remote components
  - Remote servers broadcasting to all users (e.g. scheduled downtime)
Data Preparation

- Messages containing data types cannot simply be sent
- Data must first be converted into a byte array
- This process is called serialization, or marshaling
Communication Models

- Remote Method Invocation (RMI)
- Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP)
- User Datagram Protocol (UDP)
- TCP Streams
  - Data Stream
  - Object Stream
- Piped Input/Output Streams
Remote Method Invocation (RMI)

- Allows communication using sockets
- High level interface provided by the Java System Developer Kit (JDK)
- Allows for transparency between a local method call and a remote method call
- TCP method - acknowledgement of delivery
Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP)

- Extension of Remote Method Invocation (RMI)
- Adds the Common Object Request Broker Architecture (CORBA)
- Allows for communication between components regardless of:
  - Environment
  - Programming language
User Datagram Protocol (UDP)

- Connectionless transport increases performance
  - However, no acknowledgements

- Java has built in support for UDP programming
TCP Streams

- **Data Streams**
  - Allow objects to communicate using byte arrays

- **Object Streams**
  - Allows encapsulation of entire message into an object to be sent
Piped Input/Output Streams

- Two processes can be directly connected
- Output stream of one process is connected to the input stream of another process
- High performance
- Not distributable
Performance Comparisons

- RMI vs RMI-IIOP

- RMI’s performance compared to:
  - Piped Streams
  - UDP
  - TCP Streams
Matjaz Juric et al. developed a study to test the performance of RMI and its counterpart, RMI-IIOP.

First performance test of its kind

Results focused on three areas:
- Round Trip Time (RTT)
- Network Throughput
- Performance Degradation
Test Specifications

- Two remote method types
  - Method with parameters and no return value
  - Method with no parameters and a return value

- Up to eight concurrent clients

- Different data types and array of data types

- Each measurement repeated 20 times and each method was invoked 200 times
Test #1

- Calculating the RTT of different data types used as parameters

- Single client

- Basic data types such as int, float, char, etc.

- Custom data types such as objects and structs
Results

Juric et al. “Java 2 Distributed Object Models Performance Analysis, Comparison and Optimization” [2]
Test #2

- Array of data types was used ranging from 1 to 16,384 elements

- Network throughput tested for larger arrays
Results

- Overall results were the same – RMI’s RTT was lower

- Network throughput had a different outcome
  - RMI peak throughput: 37.4 Mbps
  - RMI-IIOP peak throughput: 18.4 Mbps
Test #3

- Up to eight concurrent clients
- Each request executed in a thread to prevent blocking
- Once again, RMI outperformed RMI-IIOP
Updated Tests

- Initial tests used a beta version of RMI-IIOP

- Tests were repeated with builds 7B, 8C, FCS2B, and the final release build of RMI-IIOP
Results

Juric et al. “Java 2 Distributed Object Models Performance Analysis, Comparison and Optimization” [2]
Mark Pendergast tested the performance of RMI and other methods in a 2011 study.

All tests were performed on a single machine running two JVMs.

Two types of methods used:
- Unidirectional: With parameters and no return type
- Bidirectional: Remote method returned parameters received
Unidirectional Results

Pendergast, “Performance, overhead, and packetization characteristics of Java application level protocols” [3]
Bidirectional Results

Pendergast, “Performance, overhead, and packetization characteristics of Java application level protocols” [3]
Results

- Piped Streams was the best performance
  - Not distributed

- UDP was the second fastest option
  - No acknowledgements

- TCP DataStream and RMI were the next fastest
  - Performance decrease from acknowledgement overhead
Results - Continued

- UDP is the best option for a distributed architecture
  - No acknowledgement of delivery
  - Safeguards must be set to resend message if needed

- RMI is not the fastest option
  - Provides acknowledgements
  - High level interface makes development easy
Summary

- Efficient communication is key to distributed systems
- No single or best method
  - All have benefits and downsides
- RMI is constantly updated to improve performance
References


