Topics

- Goals
- Important Benchmarking Issues & Infrastructure
- Speed and End-to-End Latency
- Effect of QoS Settings
- Determinism / Predictability
- Throughput
- Footprint
- Scalability
- RT CORBA Priorities and Priority Inversion
Goal of this Presentation

- Examine ORB benchmarking in light of Embedded and Real-Time concerns
- Propose additional measures and relate them to ORB characteristics
- Look at the issues related to measuring certain aspects of an ORB

But:
- *Not* to compare any specific ORB’s performance
- *Not* to focus on the details of any one kind of benchmark – rather to survey many
- *Not* to focus on any one platform’s specific issues
- *Not* to focus on issues related to time measurements on multiple machines

Important Benchmarking Issues

- The general purpose of running a benchmark is to determine the fitness of the ORB
  - Either pass/fail or raking of multiple ORBs
- There are many qualities that can/should be measured
  - All benchmarks that are run need to be related to a real-time goal or quality – otherwise, why measure them
- Once something is measured, if it isn’t satisfactory, the next big question is “why is it this way?”
  - Is it the ORB? The Hardware? The OS? The TCP Stack (or replacement transport)?
- Isolating the cause is difficult
Benchmarking Infrastructure

◆ Timers
  - Need for High Resolution
    - E.g., on PPC/VxWorks, the time is limited to 17 milliseconds
    - Clearly this is of no use for real-time ORBs
  - Platform specific nature
    - The supplied resolution varies with OS and HW
  - OIS approach for dealing with this was to write our own high resolution timer where needed and wrap inside a portable interface
    - YMMV

◆ Things that help:
  - Stop/Start, Timer Overhead, Number of Trials
  - Calculation of min, max, average, std. Deviation
  - Friendly Output (e.g., .csv)

Speed

◆ Theoretically Real-Time systems are about more than just speed
  - Pragmatically that’s the first, most important, thing everyone wants to know about

◆ Multiple Dimensions of Speed
  - Simple: End-to-end latency of two way call
    - A single data point
  - Moderate: multiple calls with varying parameter sizes
    - A two dimensional set of points
  - Complex: multiple size data and varying data types
    - 3 dimensional set of data
  - 4+ dimensions could include: one-way vs two-way; varying numbers of target objects and operations, etc.
Speed [2]

- Need to make it easy for users to measure more than just small data two-ways
  - Minimally need to measure end-to-end round trip latency for a variety of data sizes

- Need to ensure that the End-to-End round trip time is divided between the Net and Gross times
  - Must be easy to measure the corresponding socket times and computer the “Net” overhead

---

End-to-End Latency for Varying Sized Parameters
End-to-End Latency for Varying Sized Parameters [2]

![Graph showing latency vs data size for different ORBs and Sockets.]

Effect of QoS Settings on Transport Benchmarks

- Another dimension to most of the previous benchmarks listed
- Goal: Determine the ability of the ORB to allow the transport’s QoS settings to be used to affect performance, determinism, etc.
- Involves repeating many of the previous tests using multiple QoS settings
- Example important TCP QoS values:
  - Send/Receive Buf Size
  - TCP No Delay
- Non TCP transports will have different QoS settings
Latency Summary

- **What we know:**
  - Round trip time for a given data size
  - General “ORB overhead” information
  - Ability to compare net values for multiple ORBs

- **What we don’t know**
  - Sources of any inefficiencies – the ORB, the OS, the Hardware, the TCP Stack, etc.
  - How the ORB will react on a different OS or Hardware Platform

Determinism / Predictability

- **Hard Real-Time Systems** worry about maximum value

- **Soft Real-Time Systems** worry about average value and distribution of values

- **Look at the distribution of the latency over a number of runs**
  - Will *not* see a normal (bell shaped) distribution
  - There is a minimum bound of the latency
  - There is *no* maximum bound
    - So the distribution is skewed toward the min and trails off exponentially toward the max
Determinism / Predictability [2]

- Multiple possible contributors to the variance of the runs:
  - Initial connection vs. existing connection
  - Other processing on the computer
  - Other traffic on the network
  - Sources of non-determinism in the ORB
  - Sources of non-determinism in the OS
  - Sources of non-determinism in the Network Stack

- How can these be determined/isolated?

- How to assess impact of these on the ORB’s use on the project?
  - One thing to do is to run many different kinds of tests

---

Determinism Comparison
Small Data Two Ways

![Graph showing latency (µs) vs. bytes for different systems and configurations.](image-url)
Determinism Conclusions

- Don’t expect predictability on a non-real-time OS
- Don’t expect predictability using TCP/IP
- Remember that no system can be predictable with all things changing
- Isolating each variable helps to understand effects
Throughput

- Inverse of latency (mostly)
- Goal: This test is important to embedded and real-time systems that transfer large amounts of data
  - No sense having a big pipe between boxes if you can’t make use of most/all of it
- Significant differentiator between ORBs
  - Easy for an ORB to send small amounts of data very fast. Things like copies, seize/release, system calls, etc. don’t amount to much
  - But… when large amounts of data are concerned then the impact of many small mistakes can be seen
    - And throughput suffers

---

Throughput [2]

Double Sequence

```
MegaBits
0  10  20  30  40  50  60  70  80
0  10  20  30  40  50  60  70  80

Mbps
0  10  20  30  40  50  60  70  80
0  10  20  30  40  50  60  70  80

Double Sequence
```
Footprint: Static Baseline

- Not important to everyone, but critical to some
- Goal: To determine:
  - How much memory does ORB leave for application when put on the embedded board
  - ORB won’t outgrow the board under certain circumstances
- Need to examine in memory footprint of:
  - Basic ORB library — may involve multiple configurations
  - Simple Client or Server
  - Realistic Application Client or Server
    - Need to separate out ORB overhead from application footprint
- Need to isolate the influence of:
  - ORB vs. application vs. OS & network stack
  - Processor architecture & compiler (and its switches)

Scalability of the ORB [1]

- Goal: Examine how the ORB changes (e.g., memory utilization or latency) under multiple circumstances
- Two major variables:
  - Things that one changes in the Server
    - Number of Objects/Implementations
    - Number of POAs
    - Number of “Worker” Threads
  - Things that one changes in the Client
    - Number of Connections to each Server and in Total
    - Number of threads in each client
    - Distribution of clients processes to same/different machines
    - Number of Connections and Priority Bands
  - These interact with each other – changes in client behavior produce effects in the server (e.g. footprint) and vice-versa
Scalability of the ORB [2]

- Need a realistic usage scenario to measure against
- Each footprint test requires a metric that measures
  - The memory size at multiple points and derives the incremental change
  - The rate of change per unit of growth (e.g., bytes/connection)
- Also look at latency tests to determine change of latency as application scales

RT CORBA vs. non RT CORBA

- Real-time predictability vs. raw performance
  - Predictability has a price
    - Additional latency: managing thread priorities in the OS
    - Additional memory: connections for priority bands
  - Bounded behavior vs. Consistent behavior
    - Bounded behavior
      - Can analyze the worst case
      - Changes in lower priority activities does not cause higher priority to violate their bounds
    - Consistent behavior
      - Execution results in the same behavior each time
      - More stringent than bounded behavior
      - The same activity executes in the same time assuming that other activity do not compete with that activity
RT CORBA Priority Transmission

- Measuring Priority Inversion

- End-to-End correctness depends on there being no priority inversions anywhere in the chain:
  - Application
  - OS
  - ORB
  - TCP/IP Stack

- If any of the pieces do not work correctly, the result will contain inversions
  - Determining which piece is at fault may be non trivial and may require testing multiple times on multiple environments

- Graph of multiple threads at different priorities
  - Horserace chart

Multiple Thread Priority Test

[1]

Good ORB on a Bad (Non RT) OS

![Graph of multiple threads at different priorities](image)
Conclusions

- Examined real-time and embedded issues related to
  - Latency, Determinism, Plugin transports, Throughput, Scalability, Priority Propagation and Priority Inversion

- Illustrated some practical techniques for benchmarking Real-Time and embedded ORBA
  - Inspired by work done by:
    - The ORBexpress product development team
    - The benchmarking work done by Boeing’s Phantom Works group on their DII COE sponsored RT CORBA benchmarking studies
    - The internal benchmarking work done by Gautam Thaker of LM
    - And by mentoring work done in conjunction with our customers

- Need to continue to evolve additional Real-Time and Embedded CORBA specific benchmarks
  - Discuss at http://www.realtime-corba.com/