High-performance and Real-time CORBA

Douglas C. Schmidt
schmidt@cs.wustl.edu

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

This work was sponsored in part by Boeing and DARPA
Many applications require high-performance

- e.g., telecom, imaging, WWW

Building these applications manually is hard

Existing middleware doesn’t support performance effectively

- e.g., CORBA, DCOM, DCE
Candidate Solution: CORBA

Goals of CORBA

- Simplify distribution by automating
  * Object location & activation
  * Parameter marshaling
  * Demultiplexing
  * Error handling
- Provide foundation for higher-level services

Goals of CORBA

- Simplify distribution by automating
  * Object location & activation
  * Parameter marshaling
  * Demultiplexing
  * Error handling
- Provide foundation for higher-level services

www.cs.wustl.edu/~schmidt/corba.html
Performance Challenges for ORB Middleware

Key Challenges
- Specifying QoS requirements
- Determining operation schedules
- Alleviating priority inversion and non-determinism
- Reducing latency/jitter for demultiplexing
- Reducing presentation layer overhead
- Maintaining small footprint
The ACE ORB (TAO)

- **TAO Overview**
  - A real-time, high-performance ORB
  - Leverages ACE
    * Runs on POSIX, Win32, RTOSs

- **Related work**
  - U. RI, Mitre
  - QuO at BBN

www.cs.wustl.edu/~schmidt/TAO.html
The ADAPTIVE Communication Environment (ACE)

- ACE Overview
  - Concurrent OO networking framework
  - Ported to C++ and Java
  - Runs on RTOSs, POSIX, and Win32

- Related work
  - x-Kernel
  - SysV STREAMS

www.cs.wustl.edu/~schmidt/ACE.html
Scope: Performance Optimizations in TAO
Problem: Providing QoS to CORBA Operations

- **Design Challenges**
  - Specifying/enforcing QoS requirements
  - Focus on *Operations* upon *Objects*
    - Rather than on communication channels or threads/synchronization
  - Support static *and* dynamic scheduling

- **Solution Approach**
  - Servants publish resource (*e.g.*, CPU) requirements and (periodic) deadlines
  - Most clients are also servants

```c
struct RT_Info {
    Time worstcase_exec_time_;  // worst-case execution time
    Period period_;            // period
    Criticality criticality_;  // criticality
    Importance importance_;   // importance
};
```
Solution: TAO’s Real-time Static Scheduling Service

```
struct RT_Info {
    Time worstcase_exec_time_;  
    Period period_;            
    Criticality criticality_;  
    Importance importance_;   
};
```

1: CONSTRUCT CALL CHAINS OF RT_OPERATIONS
2: IDENTIFY THREADS
3: POPULATE RT_INFO REPOSITORY
4: ASSESS SCHEDULABILITY
5: ASSIGN OS THREAD PRIORITIES AND DISPATCH QUEUE ORDERING SUBPRIORITIES
6: SUPPLY PRIORITIES TO ORB

OFF-LINE SCHEDULER

RT_INFO REPOSITORY

MODE 0
MODE 1
MODE 2
MODE 3

CURRENT MODE SELECTOR

Priority/Subpriority Table Per Mode

RUN-TIME SCHEDULER

I/O SUBSYSTEM

ORB CORE

OBJECT ADAPTER

RT Operation
RT Operation
RT Operation

DEPENDS UPON = EXECUTES AFTER

www.cs.wustl.edu/~schmidt/TAO.ps.gz
**Solution Approach**

- Integrate RT dispatcher into ORB endsystem
- Support multiple request scheduling strategies
  * e.g., RMS, EDF, and MUF
- Requests ordered across thread priorities by OS dispatcher
- Requests ordered within priorities based on data dependencies and importance
ORB Latency and Priority Inversion Experiments

- Vary ORBs, hold OS constant
- Solaris real-time threads
- High priority client $C_0$ connects to servant $S_0$ with matching priorities
- Clients $C_1 \ldots C_n$ have same lower priority
- Clients $C_1 \ldots C_n$ connect to servant $S_1$
- Clients invoke twoway CORBA calls that cube a number on the servant and returns result

Douglas C. Schmidt  
High-performance CORBA Tutorial

Washington University, St. Louis
• Synopsis of results
  – TAO’s latency is lowest for large # of clients
  – TAO avoids priority inversion
    * i.e., high priority client always has lowest latency
  – Primary overhead stems from concurrency and connection architecture
    * e.g., synchronization and context switching
ORB Jitter Results

- **Definition**
  - Standard deviation from average latency

- **Synopsis of results**
  - TAO’s jitter is lowest and most consistent
  - CORBAplus’ jitter is highest and most variable
Problem: Improper ORB Concurrency Model

- Common Problems
  - High overhead
    * Context switching
    * Synchronization
  - Thread-level priority inversions
    * FIFO request queueing
    * Improper thread priorities
  - Lack of application control over concurrency model
Problem: ORB Shared Connection Model

- Common Problems
  - Request-level priority inversions
    * Sharing multiple priorities on a single connection
  - Complex connection multiplexing
  - Synchronization overhead
Problem: High Locking Overhead

- Locking overhead significantly affects latency and jitter
  - Memory management commonly involves locking
- RT ORBs should minimize or eliminate all locking operations
- TAO is carefully designed to minimize locking and memory allocation
Solution: TAO’s Inter-ORB Connection Topology

www.cs.wustl.edu/~schmidt/RT-middleware.ps.gz
Real-time OS/ORB Performance Experiments

- Vary OS, hold ORBs constant
- Single-processor Intel Pentium Pro 200 Mhz, 128 Mbytes of RAM
- Client and servant run on the same machine
- Client $C_i$ connects to servant $S_i$ with priority $P_i$
  - $i$ ranges from 1...50
- Clients invoke twoway CORBA calls that cube a number on the servant and returns result

Pentium Pro

www.cs.wustl.edu/~schmidt/RT-OS.ps.gz
Real-time OS/ORB Performance Results

- LynxOS yielded very good latency and deterministic behavior
- Erratic behavior and high latency are a problem for Windows NT
- Windows NT also showed priority inversion at 50 low priority clients
- VxWorks performs surprisingly erratically
- Solaris’ latency is high but predictable
Real-time OS/ORB Jitter Results

- **Definition**
  - Standard deviation from average latency

- **Synopsis of results**
  - Some RTOS’s provide low jitter
  - ORB (TAO) doesn’t introduce jitter
Problem: Reducing Demultiplexing Latency

- **Design Challenges**
  - Minimize demuxing layers
  - Provide \(O(1)\) operation demuxing
  - Avoid priority inversions
  - Remain CORBA-compliant
Solution: Demultiplexing Optimizations

- Results at www.cs.wustl.edu/~schmidt/ieee_tc-97.ps.gz
- Linear search based on Orbix demuxing strategy
- Perfect hashing based on GNU gperf
  - www.cs.wustl.edu/~schmidt/gperf.ps.gz
Demultiplexing Performance Results

- **Synopsis**
  - Linear search is far too costly
  - Dynamic hashing can be unstable
  - `gperf` solution is 100% compatible, but static
  - Optimal active demuxing may not be 100% compatible, but is dynamic
  - Strategy pattern facilitates flexibility
Next Steps: Integrating TAO with ATM I/O Subsystem

- Key Features
  - Vertical integration of QoS through ORB, OS, and ATM network
  - Real-time I/O enhancements to Solaris kernel
  - Provides rate-based QoS end-to-end
  - Leverages APIC features for cell pacing and zero-copy buffering
Next Steps: Strategized Scheduling Service Framework

1. SPECIFY RT_OPERATION CHARACTERISTICS AND DEPENDENCIES
2. POPULATE RT_INFO REPOSITORY
3. ASSESS SCHEDULABILITY
4. ASSIGN STATIC PRIORITY AND STATIC SUBPRIORITY
5. MAP STATIC PRIORITY, DYNAMIC SUBPRIORITY, AND STATIC SUBPRIORITY INTO DISPATCHING PRIORITY AND DISPATCHING SUBPRIORITY
6. ASSIGN DISPATCHING QUEUE CONFIGURATION
7. SUPPLY DISPATCHING QUEUE CONFIGURATION TO THE ORB
8. CONFIGURE QUEUES BASED ON DISPATCHING QUEUE CONFIGURATION
9. SUPPLY STATIC PORTIONS OF DISPATCHING PRIORITY AND DISPATCHING SUBPRIORITY TO THE ORB
10. DYNAMIC QUEUES ASSIGN DYNAMIC PORTIONS OF DISPATCHING SUBPRIORITY (AND POSSIBLY DISPATCHING PRIORITY)

www.cs.wustl.edu/~schmidt/dynamic.ps.gz
Principles for High-Performance, Real-time ORBs

- Avoid dynamic connection management
- Minimize dynamic memory management and data copying
- Avoid multiplexing connections for different priority threads
- Avoid complex concurrency models
- Integrate ORB with OS and I/O subsystem and avoid reimplementing OS mechanisms
- Guide ORB design by empirical benchmarks
TAO Project Research Summary

- **Current focus: real-time ORBs**
  - Developed first deployed real-time CORBA scheduling service and first POA
  - Minimized ORB Core priority inversion and non-determinism
  - Reduced end-to-end latency via demuxing optimizations
  - Applied optimizations to IIOP protocol engine
  - Co-submitters to OMG’s real-time CORBA RFP

- **Future work**
  - Dynamic and hybrid scheduling of CORBA operations
  - Distributed QoS and integration with real-time ATM I/O Subsystem
  - Optimizing IDL compiler
  - Technology transfer with DARPA Quorum program
Web URLs for Additional Information

- **These slides:**
  www.cs.wustl.edu/~schmidt/PDF/tutorial4.ps.gz

- **More information on TAO:**
  www.cs.wustl.edu/~schmidt/RT-ORB.ps.gz

- **TAO Event Channel:**
  www.cs.wustl.edu/~schmidt/JSAC-98.ps.gz

- **TAO static scheduling:**
  www.cs.wustl.edu/~schmidt/TAO.ps.gz

- **TAO dynamic scheduling:**
  www.cs.wustl.edu/~schmidt/dynamic.ps.gz

- **ORB Endsystem Architecture:**
  www.cs.wustl.edu/~schmidt/RT-middleware.ps.gz
Web URLs for Additional Information (cont’d)

- **Performance Measurements:**
  - DII throughput: [www.cs.wustl.edu/~schmidt/GLOBECOM-96.ps.gz](http://www.cs.wustl.edu/~schmidt/GLOBECOM-96.ps.gz)

- **More detail on CORBA:** [www.cs.wustl.edu/~schmidt/corba.html](http://www.cs.wustl.edu/~schmidt/corba.html)

- **ADAPTIVE Communication Environment (ACE):**
  [www.cs.wustl.edu/~schmidt/ACE.html](http://www.cs.wustl.edu/~schmidt/ACE.html)

- **The ACE ORB (TAO):**
  [www.cs.wustl.edu/~schmidt/TAO.html](http://www.cs.wustl.edu/~schmidt/TAO.html)