Patterns and Performance of Real-time Middleware for Embedded Systems

Douglas C. Schmidt

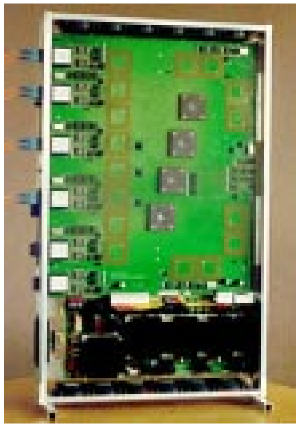
Associate Professor &
Director of the Center for
Distributed Object Computing

Computer Science Dept.
Washington University, St. Louis
www.cs.wustl.edu/∼schmidt/

Lockheed Martin

November 1^{st}, 1999

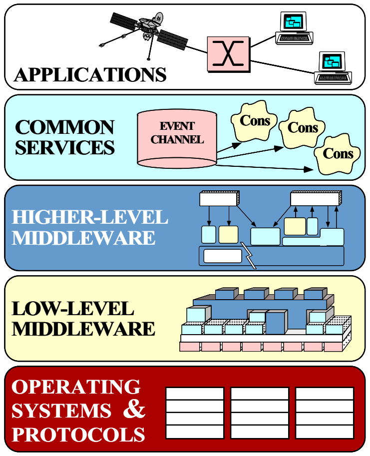
Motivation: the QoS-enabled Software Crisis



www.arl.wustl.edu/arl/

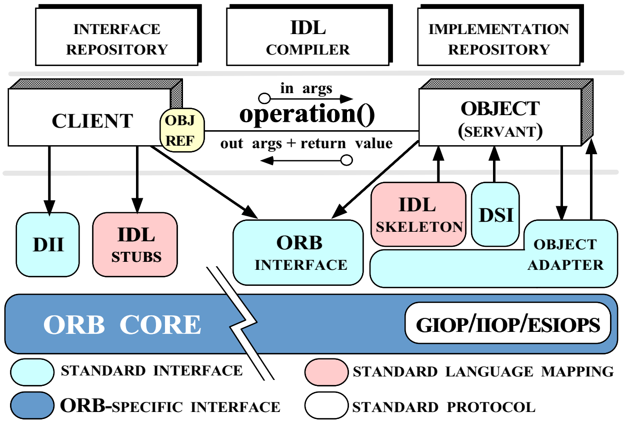
- **Symptoms**
  - Communication ***hardware*** gets smaller, faster, cheaper
  - Communication ***software*** gets larger, slower, more expensive
- **Culprits**
  - ***Inherent*** and ***accidental*** complexity
- **Solution Approach**
  - ***Standard communication middleware***

Problem: Lack of QoS-enabled Middleware



- Many applications require QoS guarantees
  - *e.g.*, avionics, telecom, WWW, medical, high-energy physics
- Building these applications manually is hard
- Existing middleware doesn't support QoS effectively
  - *e.g.*, CORBA, DCOM, DCE, Java
- Solutions must be integrated horizontally & vertically

Candidate Solution: CORBA

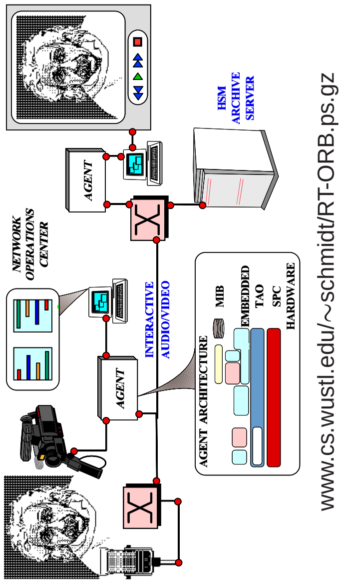


www.cs.wustl.edu/∼schmidt/corba.html

Goals of CORBA

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

Caveat: Requirements/Limitations of CORBA for QoS-enabled Systems



www.cs.wustl.edu/∼schmidt/RT-ORB.ps.gz

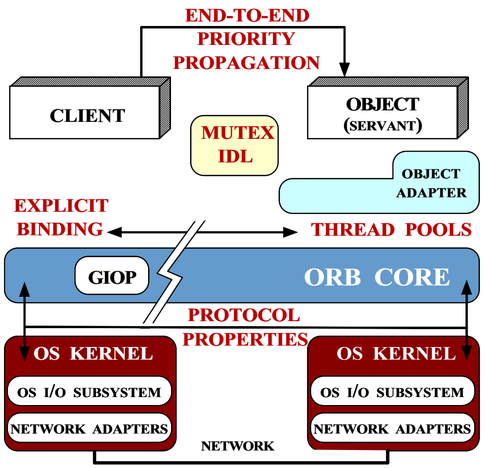
Requirements

- Location transparency
- Performance transparency
- Predictability transparency
- Reliability tranparency

Limitations

- Lack of QoS specifications
- Lack of QoS enforcement
- Lack of real-time programming features
- Lack of performance optimizations

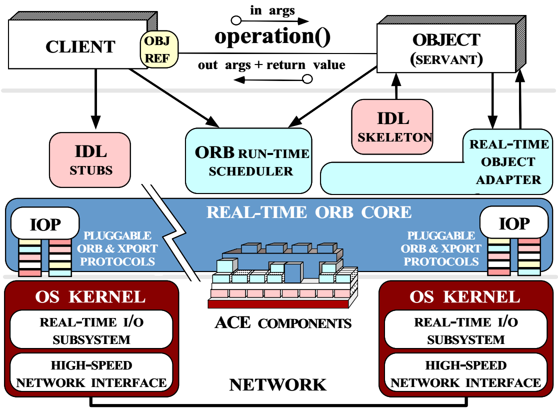
Overview of the Real-time CORBA Specification



Features

1. End-to-end priority propagation
2. Protocol properties
3. Thread pools
4. Explicit binding
5. Mutex IDL

Our Approach: The ACE ORB (TAO)

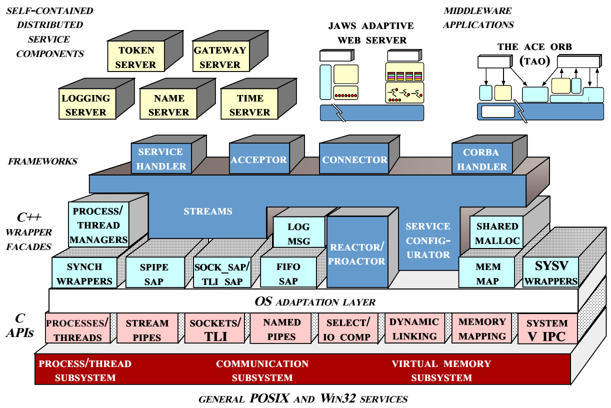


www.cs.wustl.edu/∼schmidt/TAO.html

TAO Overview →

- An open-source, standards-based, real-time, high-performance CORBA ORB
- Runs on POSIX, Win32, & embedded RT platforms
  - *e.g.*, VxWorks, Chorus, LynxOS
- Leverages ACE

The ADAPTIVE Communication Environment (ACE)



www.cs.wustl.edu/∼schmidt/ACE.html

ACE Overview →

- A concurrent OO networking framework
- Available in C++ and Java
- Ported to POSIX, Win32, and RTOSs

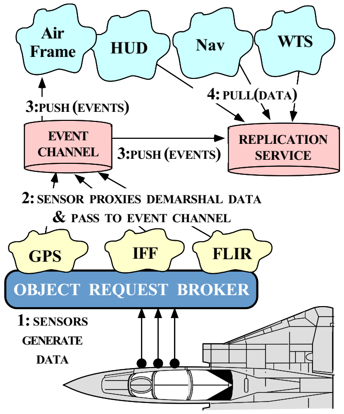
Related work →

- x-Kernel
- SysV STREAMS

ACE and TAO Statistics

- Over 35 person-years of effort
  - ACE > 200,000 LOC
  - TAO > 125,000 LOC
  - TAO IDL compiler > 100,000 LOC
  - TAO CORBA Object Services > 150,000 LOC
- Ported to UNIX, Win32, MVS, and RTOS platforms
- Large user community
  - www.cs.wustl.edu/~schmidt/ACE-users.html
- Currently used by dozens of companies
  - Bellcore, Boeing, Ericsson, Kodak, Lockheed, Lucent, Motorola, Nokia, Nortel, Raytheon, SAIC, Siemens, etc.
- Supported commercially
  - ACE → www.riverace.com
  - TAO → www.ociweb.com

Applying TAO to Avionics Mission Computing



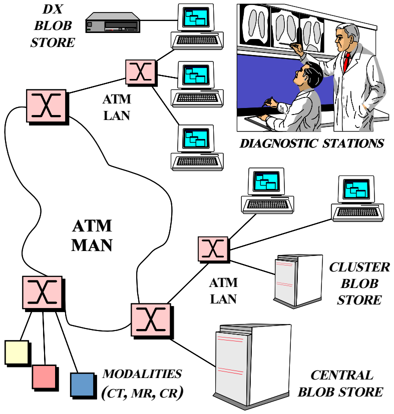
www.cs.wustl.edu/~schmidt/JSAC-98.ps.gz

Domain Challenges

- Deterministic & statistical real-time deadlines
- Periodic & aperiodic processing
- COTS and open systems
- Reusable components
- Support platform upgrades

www.cs.wustl.edu/~schmidt/TAO-boeing.html

Problem: Optimizing Complex Software



www.cs.wustl.edu/~schmidt/JSAC-99.ps.gz

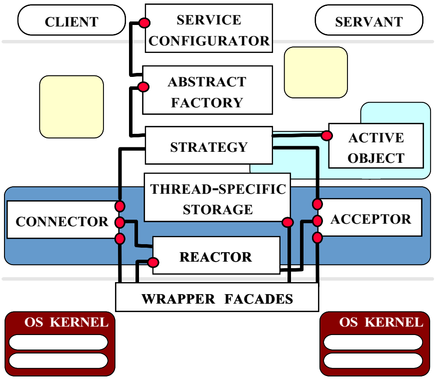
Common Problems →

- Optimizing complex software is hard
- Small “mistakes” can be costly

Solution Approach (Iterative) →

- Pinpoint overhead via *white-box* metrics
  - *e.g.*, `Quantify` and `VMEtro`
- Apply patterns and framework components
- Revalidate via white-box and black-box metrics

Solution 1: Patterns and Framework Components



www.cs.wustl.edu/~schmidt/ORB-patterns.ps.gz

Definitions

- *Pattern*
  - A solution to a problem in a context
- *Framework*
  - A “semi-complete” application built with components
- *Components*
  - Self-contained, “pluggable” ADTs

Solution 2: ORB Optimization Principle Patterns

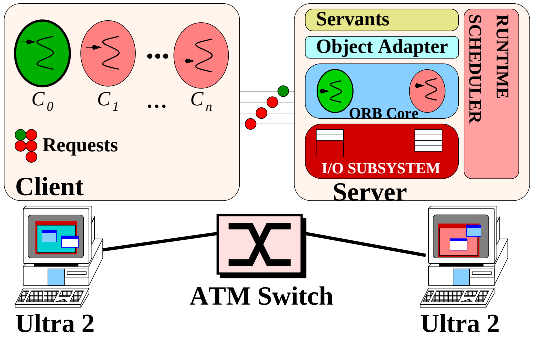
Definition

- ***Optimization principle patterns*** document rules for avoiding common design and implementation problems that can degrade the performance, scalability, and predictability of complex systems

Key Principle Patterns Used in TAO

#	**Principle Pattern**
1	Optimize for the common case
2	Remove gratuitous waste
3	Replace inefficient general-purpose functions with efficient special-purpose ones
4	Shift computation in time, *e.g.*, precompute
5	Store redundant state to speed-up expensive operations
6	Pass hints between layers and components
7	Don't be tied to reference implementations/models
8	Use efficient/predictable data structures

ORB Latency and Priority Inversion Experiments



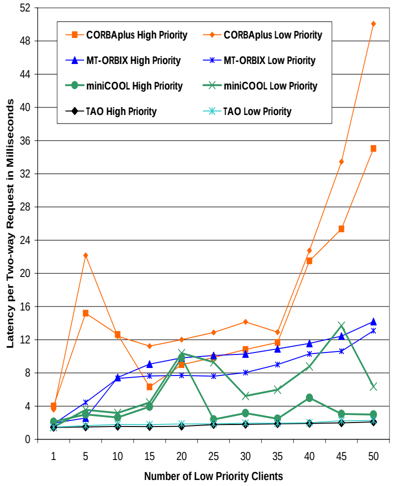
www.cs.wustl.edu/∼schmidt/RT-perf.ps.gz

Method

- 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 two-way CORBA calls that cube a number on the servant and returns result

---

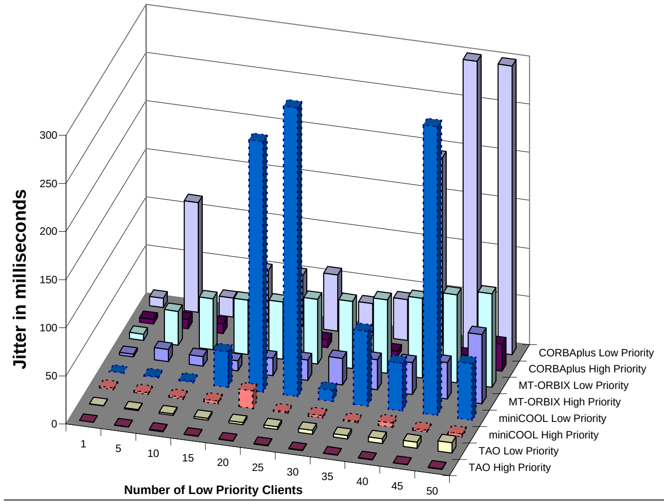
ORB Latency and Priority Inversion Results



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



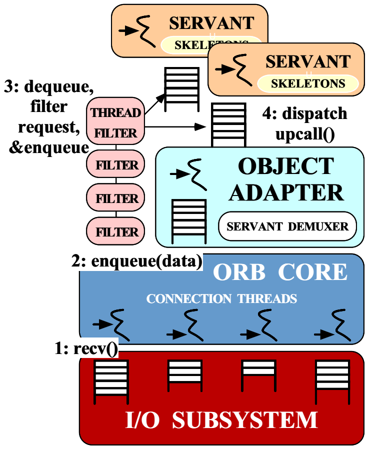
Definition

- Jitter → 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 Models

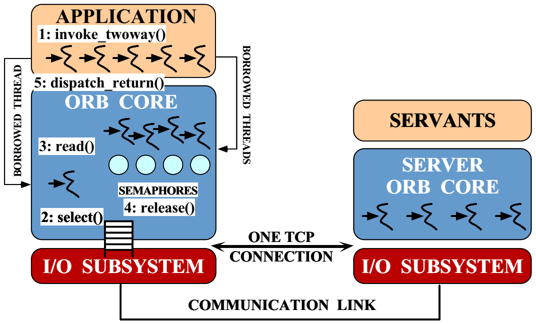


Common Problems

- High context switching and synchronization overhead
- Thread-level and packet-level priority inversions
- Lack of application control over concurrency model

www.cs.wustl.edu/∼schmidt/
CACM-arch.ps.gz

Problem: ORB Shared Connection Models

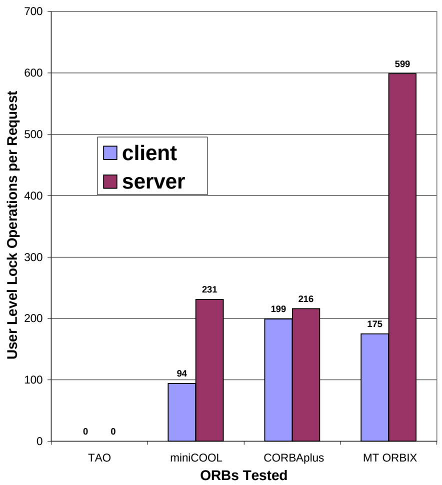


Common Problems

- Request-level priority inversions
  - Sharing multiple priorities on a single connection
- Complex connection multiplexing
- Synchronization overhead

www.cs.wustl.edu/∼schmidt/
RTAS-98.ps.gz

Problem: High Locking Overhead

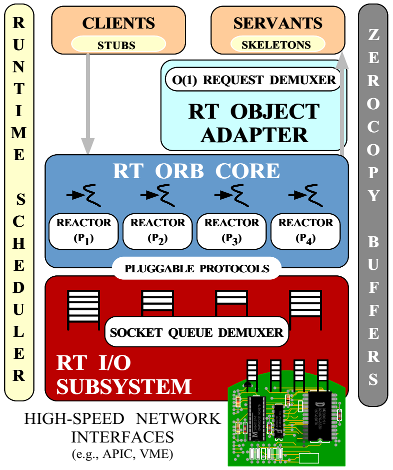


Common Problems

- Locking overhead affects latency and jitter significantly
- Memory management commonly involves locking

www.cs.wustl.edu/∼schmidt/
RTAS-98.ps.gz

Solution: TAO’s ORB Endsystem Architecture



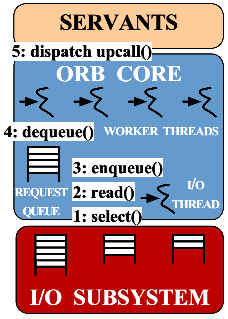
Solution Approach →

- Integrate scheduler into ORB endsystem
- Co-schedule threads
- Leader/followers thread pool

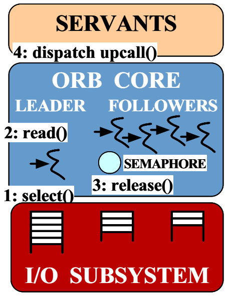
Principle Patterns →

- Pass hints, precompute, optimize common case, remove gratuitous waste, store state, don’t be tied to reference implementations & models

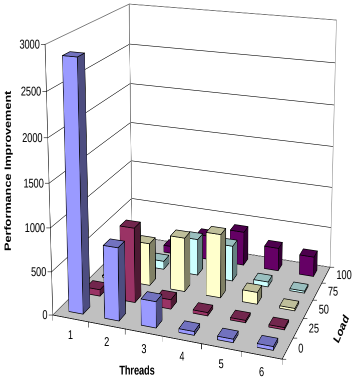
Thread Pool Comparison Results



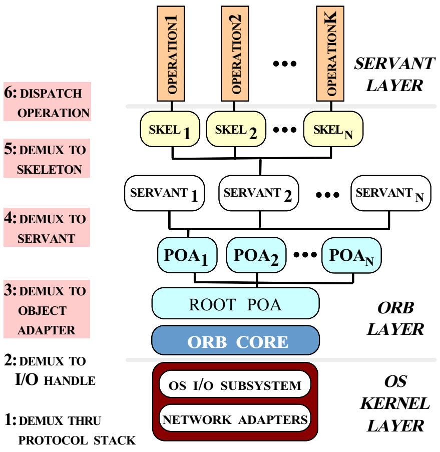
Worker Thread Pool



Leader/Follower Thread Pool



Problem: Reducing Demultiplexing Latency

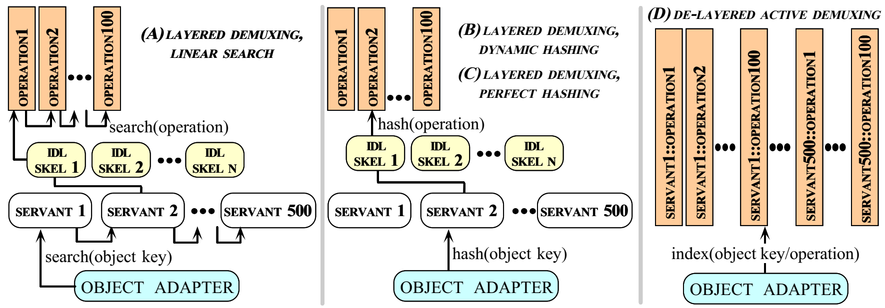


Design Challenges

- Minimize demuxing layers
- Provide $O(1)$ operation demuxing through all layers
- Avoid priority inversions
- Remain CORBA-compliant

www.cs.wustl.edu/∼schmidt/ POA.ps.gz

Solution: TAO's Request Demultiplexing Optimizations



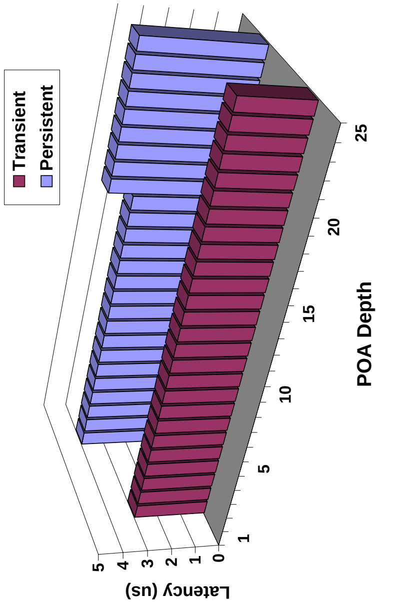
Demuxing

- www.cs.wustl.edu/∼schmidt/ {ieee_tc-97,COOTS-99}.ps.gz

Perfect hashing

- www.cs.wustl.edu/∼schmidt/ gperf.ps.gz

POA Demultiplexing Results



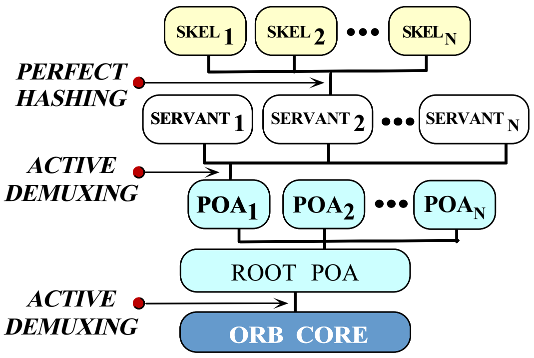
Synopsis of Results

- Active demux is efficient & predictable for both transient and persistent object references.

Principle Patterns

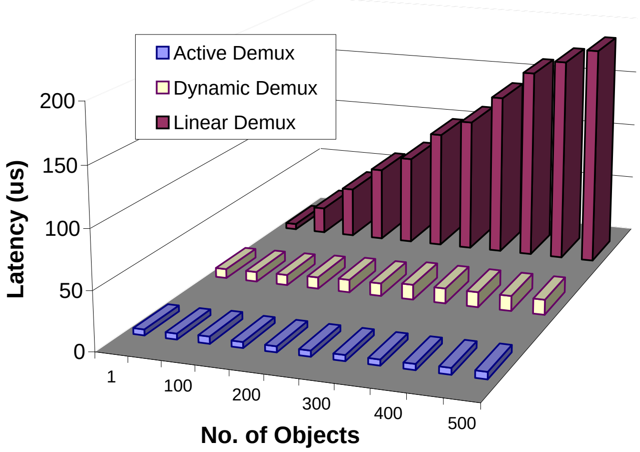
- Precompute, pass hints, use special-purpose & predictable data structures

TAO Request Demultiplexing Summary



Demultiplexing Stage	Absolute Time (μs)
1. Request parsing	2
2. POA demux	2
3. Servant demux	3
4. Operation demux	2
5. Parameter demarshaling	operation dependent
6. User upcall	servant dependent
7. Results marshaling	operation dependent

Servant Demultiplexing Results



Synopsis of Results

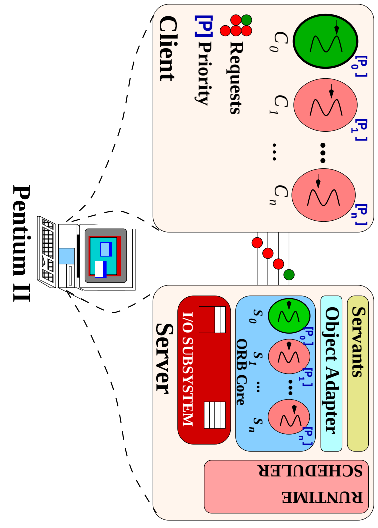
- Linear demux is costly
- Active demux is most efficient & predictable

Principle Patterns

- Precompute, pass hints, use special-purpose & predictable data structures

---

Real-time ORB/OS Performance Experiments



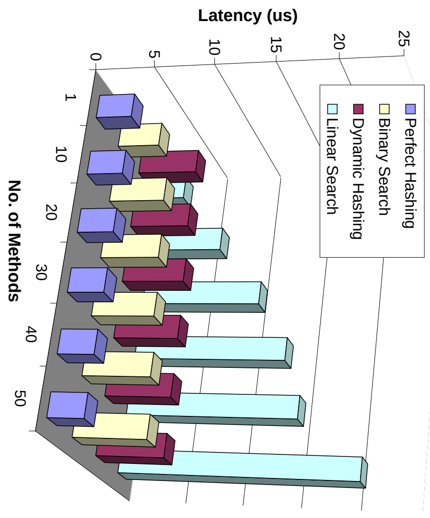
www.cs.wustl.edu/~schmidt/RT-OS.ps.gz

Method

- Vary OS, hold ORBs constant
- Single-processor Intel Pentium II 450 Mhz, 256 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 $1 \ldots 50$
- Clients invoke two-way CORBA calls that cube a number on the servant and returns result

---

Operation Demultiplexing Results



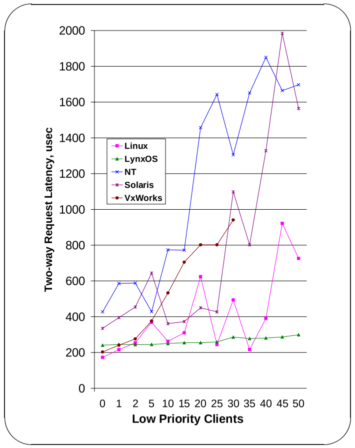
Synopsis of Results →

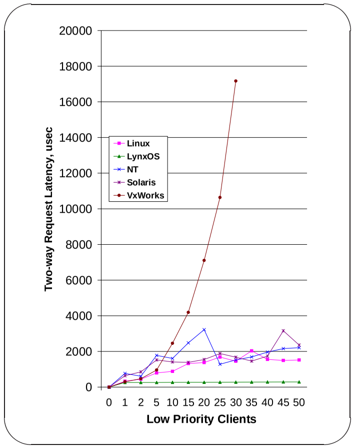
- Perfect Hashing
  - Highly predictable
  - Low-latency
- Others strategies slower

Principle Patterns →

- Precompute, use predictable data structures, remove gratuitous waste

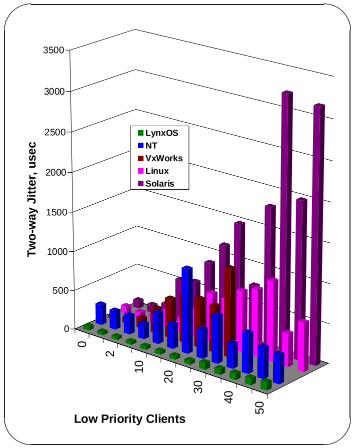
Real-time ORB/OS Performance Results

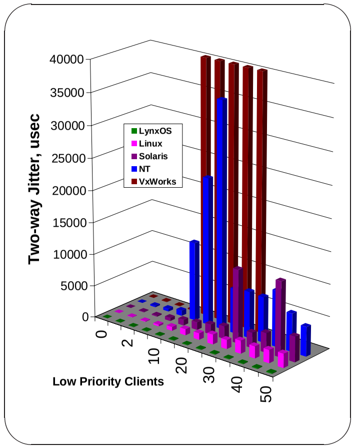




High-priority Client Latency **Low-priority Clients Latency**

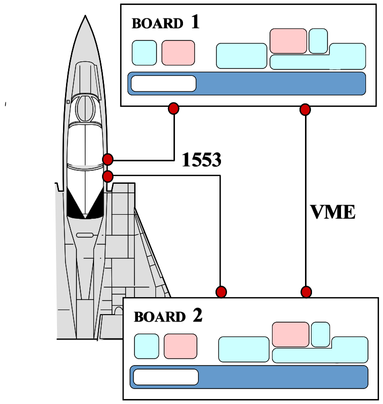
Real-time ORB/OS Jitter Results





High-priority Client Jitter **Low-priority Clients Jitter**

Problem: Hard-coded ORB Messaging and Transport Protocols



- GIOP/IIOP are not sufficient, *e.g.*:
  - GIOP message footprint may be too large
  - TCP lacks necessary QoS
  - Legacy commitments to existing protocols
- Many ORBs do not support "pluggable protocols"
  - This makes ORBs inflexible and inefficient

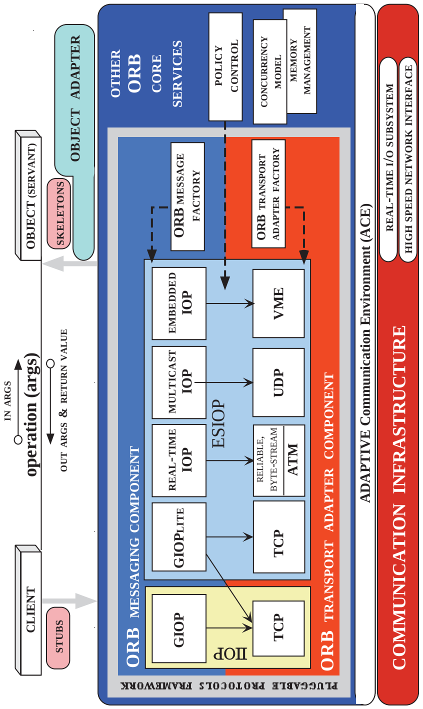
One Solution: Hacking GIOP

- GIOP requests include fields that aren't needed in homogeneous embedded applications
  - *e.g.*, GIOP magic #, GIOP version, byte order, request principal, etc.
- These fields can be omitted without any changes to the standard CORBA programming model
- TAO's `-ORBgioplite` option save 15 bytes per-request, yielding these calls-per-second:

	Marshaling-enabled			Marshaling-disabled		
	min	max	avg	min	max	avg
GIOP	2,878	2,937	2,906	2,912	2,976	2,949
GIOPlite	2,883	2,978	2,943	2,911	3,003	2,967

- The result is a measurable improvement in throughput/latency
  - However, it's so small (2%) that hacking GIOP is of minimal gain except for low-bandwidth links

Better Solution: TAO's Pluggable Protocols Framework



Features

- Pluggable *ORB messaging* and *transport* protocols
- Highly efficient and predictable behavior

Principle Patterns

- Replace general-purpose functions (protocols) with special-purpose ones

---

CORBA Protocol Interoperability Architecture

STANDARD CORBA PROGRAMMING API

ORB MESSAGING COMPONENT	GIOP	GIOPLITE	ESIOP
ORB TRANSPORT ADAPTER COMPONENT	IIOP	VME-IOP	ATM-IOP
TRANSPORT LAYER	TCP	VME DRIVER	RELIABLE SEQUENCED AAL5
NETWORK LAYER	IP		ATM

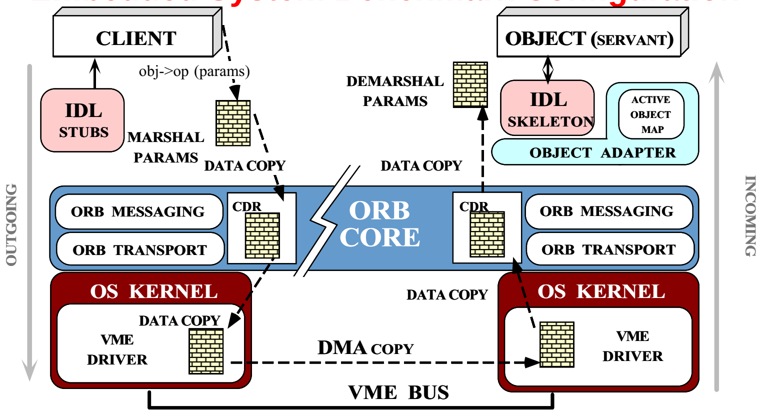
PROTOCOL CONFIGURATIONS

Features →

- Presentation layer
  - *e.g.,* CDR
- Message formats
  - *e.g.,* GIOP
- Transport assumptions
  - *e.g.,* TCP
- Object addressing
  - *e.g.,* IIOP IOR

www.cs.wustl.edu/∼schmidt/pluggable_protocols.ps.gz

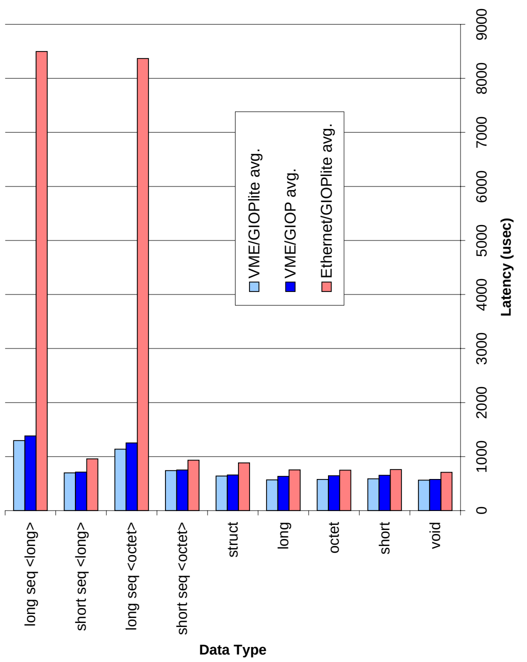
Embedded System Benchmark Configuration



VxWorks running on 200 Mhz PowerPC over a 320 Mbps VME & 10 Mbps Ethernet

---

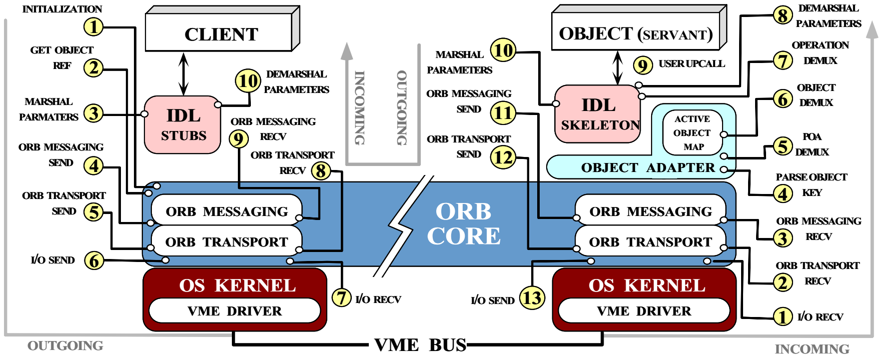
Ethernet & VME Two-way Latency Results



Synopsis of Results

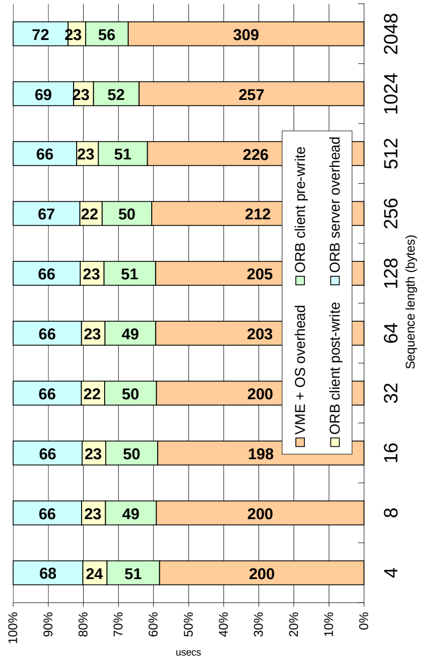
- VME protocol is much faster than Ethernet
- No application changes are required to support VME

Pinpointing ORB Overhead with VMEtro Timeprobes



- Timeprobes use VMEtro monitor, which measures end-to-end time
- Timeprobe overhead is minimal, *i.e.*, 1 μsec

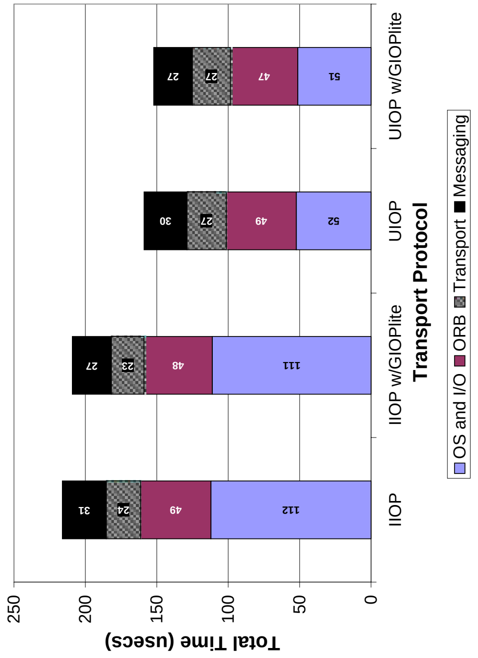
ORB & VME One-way Overhead Results



Synopsis of Results

- ORB overhead is relatively constant and low
  – *e.g.*, ~110 μsecs per end-to-end operation
- Bottleneck is VME driver and OS, not ORB

ORB & Transport Overhead Results

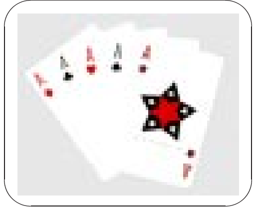


Synopsis of Results

- ORB overhead is relatively constant and low
  – *e.g.*, ~49 μsecs per two-way operation
- Bottleneck is OS and I/O operation

Lessons Learned Developing 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 and patterns



Concluding Remarks

- Researchers and developers of distributed, real-time applications confront many common challenges
  - *e.g.*, service initialization and distribution, error handling, flow control, scheduling, event demultiplexing, concurrency control, persistence, fault tolerance
- Successful researchers and developers apply *patterns*, *frameworks*, and *components* to resolve these challenges
- Careful application of patterns can yield efficient, predictable, scalable, *and* flexible middleware
  - *i.e.*, middleware performance is largely an “implementation detail”
- Next-generation ORBs will be highly QoS-enabled, though many research challenges remain

---

Web URLs for Additional Information

- Real-time CORBA 1.0 spec:
  `www.cs.wustl.edu/∼schmidt/RT-ORB-std-new.pdf.gz`
- More information on TAO:
  `www.cs.wustl.edu/∼schmidt/TAO.html`
- TAO static scheduling:
  `www.cs.wustl.edu/∼schmidt/RT-ORB.ps.gz`
- TAO dynamic scheduling:
  `www.cs.wustl.edu/∼schmidt/dynamic.ps.gz`