Static and Dynamic Scheduling in Real-Time CORBA

Chris Gill
Center for Distributed Object Computing
Department of Computer Science
Washington University, St. Louis
cdgill@cs.wustl.edu

www.cs.wustl.edu/~cdgill/research/scheduling/cs562_sched.ps.gz
www.cs.wustl.edu/~cdgill/research/scheduling/cs562_sched_4.ps.gz

21 April 1999

Research Supported in part by Boeing, USENIX, and DARPA
Contents

• Introduction
  – Hard real-time domain requirements
  – Requirements of other real-time domains
  – Scheduling terminology and algorithms

• RT-CORBA Scheduling Research
  – Hybrid static/dynamic scheduling in TAO

• Future work
  – Supporting requirements of other real-time domains
  – Empirical investigations
  – Adapting to the emerging RT-CORBA QoS framework

• Concluding remarks and further information
Hard Real-time Domain Requirements

- Support hard real-time behavior
  - Critical deadlines are met
  - Critical processing is predictable
- Utilize scarce resources efficiently
  - Conserve scarce resources
  - Minimize debugging costs
- Readily support platform (hardware/OS) upgrades
  - Ability to scale and distribute
- Reuse designs and implementations
  - Control testing, certification costs
Other Real-time Domain Requirements

- Support statistical real-time behavior
  - Meet a percentage of deadlines
  - Degrade gracefully under load
- Utilize scarce resources efficiently
  - Emphasis is often on throughput as well as latency
  - Ability to handle varying loads efficiently and fairly
- Readily support platform (hardware/OS) upgrades
- Reuse designs and implementations
QoS Specification, QoS Policy, and QoS Enforcement Components:

- **RT_Info**: Descriptor to *Specify* Real-Time Operation Information
- **Scheduling Strategy**: *Policy* for Operation Scheduling
- **Dispatching Module**: Mechanism to *Enforce* Scheduling Policy
RT_Info Descriptor

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

- **Criticality** – significance to application of operation QoS failure
- **Worst case execution time** – can be determined by techniques such as time probes, instruction counting
- **Period** – interval between invocations, in 100 nsec units
- **Importance** – tie-breaker to provide total ordering of operations
- **Dependencies** – on results produced by other operations, which must run before the operation is dispatched
Scheduling Strategy

- Platform-independent mapping
  - From operation characteristics into operation *urgency*
  - Ordered tuple: \(<\text{static priority}, \text{dynamic subpriority}, \text{static subpriority}>\)

- Platform-specific mapping
  - From operation urgency into operation preemption priority and preemption subpriority
  - Generates configuration for dispatching module
Scheduling Strategy: Rate-Monotonic

- Assigns higher priorities to operations with higher rates
- Can be completely specified prior to run-time if periods do not vary
- Can be enforced very efficiently and deterministically at run-time
- Limitation – often must over-provision the system to account for the worst case
Scheduling Strategy: Earliest Deadline

- Assigns higher priorities to operations with nearer deadlines
- Static priority is constant, dynamic ordering done at run-time
- Greater flexibility, as the operations need not be known prior to run-time
- Limitations – Higher dispatching overhead, cannot isolate deadline failures if the system is overloaded
**Scheduling Strategy: Minimum Laxity**

- Assigns higher priorities to operations with less slack.
- A refinement of earliest deadline – considers execution time.
- Can *predict* some deadline failures prior to dispatch, and avoid wasting resources on doomed operations.
- Limitations – same as earliest deadline.

---

Washington University, St. Louis


Scheduling Strategy: Maximum Urgency

- Hybrid approach with static and dynamic components
  - Assigns static priority according to operation criticality
  - Assigns dynamic subpriority according to laxity
- Offers isolation of deadline failures, with high resource utilization
- Limitation – same overhead issues as earliest deadline and minimum laxity strategies
Dispatching Modules

- Dispatching modules perform dispatch prioritization based on configuration information from the scheduling policy.
  - Thread-priority-per-queue enforces preemption priority.
  - Per-queue policy (e.g., FIFO earliest-deadline, minimum laxity) enforces preemption subpriority.
  - If subpriority policy is FIFO, may omit extra queueing.

- Can locate dispatching modules at various points in the architecture.
Progression of Scheduling Approaches

- Historic influences on TAO’s scheduling approach came largely from the hard real-time avionics domain

- Even within that domain, scheduling approaches have evolved significantly
  - Static cyclic executives were tightly coupled to the application
  - Static preemptive dispatching via RMS gave functional decoupling
  - Strategized scheduling gave QoS decoupling

- Future work will extend this evolution to systems with statistical and multi-dimensional QoS requirements
Hybrid Static/Dynamic Scheduling in TAO

- Goal: achieve higher utilization by scheduling more of the unused time
- Goal: preserve stability of the schedule under load by isolating missed deadlines to non-critical operations
- Goal: let applications specify which operations are critical
- Hypothesis: with hybrid scheduling techniques we can achieve these goals without undue overhead or schedule instability under load
Hybrid Static/Dynamic Scheduling, Cont’d

- **Solution Approach**
  - At configuration time, the scheduling strategy generates static information
    - Assigns a static (thread) priority to each operation
    - Specifies priority and sub-priority enforcement policies
  - Factories configure ORB and ORB Services components according to the specified policies
  - At run-time, configured components enforce priority and subpriority
TAO’s Strategized Scheduling Service Architecture

http://www.cs.wustl.edu/~schmidt/dynamic.ps.gz
The Cost of Dynamic Scheduling

- Solaris 2.5.1/Ultra 30, 300MHz CPU, 256 MB RAM
- Server & client on same CPU
- Real-Time Scheduling class
- One high priority (20 Hz) client, varying number of low priority (10 Hz) clients
- Small (< 10 percent) overhead for dynamic (MUF) vs. static (RMS) scheduling

**TAO Event Channel Request Latency**

Washington University, St. Louis
Future Work

- Supporting requirements of other real-time domains:
  - Web Servers, E-commerce applications
  - Require multi-dimensional QoS: latency, throughput, and fairness
  - Will build on previous work: strategized framework, strategies

- Empirical investigations:
  - Investigate effects of various priority and queue management mechanisms on end-to-end performance
  - Investigate performance of various pluggable network protocols
  - Investigate performance effects of priority, time, and fairness constraints in scheduling strategies and enforcement mechanisms

- Adapting our approach to the emerging RT-CORBA QoS framework
The RT-CORBA QoS Framework

- The CORBA Messaging Joint Revised Submission (orbos/98-05-05) defines an overall QoS framework that includes policy management for request priority, queueing, and timeouts

- The mechanisms defined in this specification are necessary to enforce static priority preservation in the ORB, and are extended in the RT-CORBA 1.0 Joint Revised Submission (orbos/98-12-10)

- The Dynamic Scheduling RFP (orbos/98-02-15) asks responders to address additional issues for dynamic scheduling

- The RT-CORBA QoS framework must be extended to support policy driven enforcement of dynamic and hybrid static/dynamic scheduling

- TAO’s strategized scheduling framework can be readily adapted to implement this extended RT-CORBA QoS framework (once the specification is adopted as part of the CORBA standard)
Policy management in the RT-CORBA QoS framework

- QoS is managed through interfaces derived from CORBA::Policy
- Each Policy has an associated PolicyType that can be queried
- A PolicyList is sequence of policies (efficient bulk transfer)
- Client-side policies are specified at three overriding levels:
  - ORB level through PolicyManager
  - Thread level through PolicyCurrent
  - Object level though Objects
- Server-side policies are specified by associating QoS policy objects with a POA (can be passed as arguments to POA::create_POA)
- QoS policies and overrides can be established and validated at client initialization through calls to Object::validate_connection
Priority management in the RT-CORBA QoS framework

- **Two priority levels**
  - RT-CORBA defines platform independent priority values from `RT_CORBA::minPriority (0)` to `RT_CORBA::maxPriority (32767)`
  - These are mapped into native thread priorities by a default `PriorityMapping`, which an application can replace

- **Priority inheritance**
  - Server can re-map priorities
  - Server can operate at fixed or at client request priority
Queue management in the RT-CORBA QoS framework

- Queue order management

  - CORBA Messaging defines queue ordering policies:
    * ORDER_ANY (don’t care, fifo)
    * ORDER_TEMPORAL (order requests were issued)
    * ORDER_PRIORITY (static)
    * ORDER_DEADLINE (dynamic)

  - Static set of policies: needs to be extensible to support new strategies
Concluding Remarks

- Hybrid static/dynamic scheduling supports higher utilization while preserving the hard real-time behavior of critical operations under load, and dynamic scheduling overhead appears sufficiently small.

- Priority must be supplemented with additional information to achieve dynamic or hybrid scheduling.

- QoS frameworks introduced with the CORBA Messaging specification can be extended to support QoS requirements of other real-time domains.

- Related work
  - BBN QuO: Zinky, Bakken, and Schantz, ’95
  - CMU RT-Mach: Lee, Yoshida, Mercer, and Rajkumar ’96
For Further Information

- **TAO Scheduling:**
  
  www.cs.wustl.edu/~schmidt/dynamic.ps.gz
  www.cs.wustl.edu/~schmidt/DASC-98.ps.gz
  www.cs.wustl.edu/~schmidt/TAO.ps.gz

- **TAO:**
  
  www.cs.wustl.edu/~schmidt/TAO.html

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

- **These slides:**
  
  www.cs.wustl.edu/~cdgill/research/scheduling/cs562_sched.ps.gz
  www.cs.wustl.edu/~cdgill/research/scheduling/cs562_sched_4.ps.gz