Static and Dynamic Scheduling using the RT-CORBA QoS Framework

Christopher D. Gill
Center for Distributed Object Computing
Washington University, St. Louis
cdgill@cs.wustl.edu

www.cs.wustl.edu/~cdgill/research/scheduling/QoS-frameworks.ps.gz

25 January 1999

Research Supported by The Boeing Company
Contents

- Hard real-time domain requirements
- Static/dynamic scheduling tradeoffs
- Current TAO scheduling service design
- Migrating to the RT-CORBA QoS framework
- Extending the RT-CORBA QoS framework for dynamic and hybrid static/dynamic scheduling
- Concluding remarks
- Future work
- For 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
Static and Dynamic Scheduling Approaches

- **Criticality** is an application-specified operation significance
- **schedulability** indicates whether there are sufficient resources to perform all critical operations
- Operations are dispatched in order of **urgency**

<table>
<thead>
<tr>
<th>Scheduling Algorithm</th>
<th>Static Priority Levels</th>
<th>Schedulability Basis</th>
<th>Urgency Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Urgency First</td>
<td>one per criticality level (at least two: high, low)</td>
<td>critical set</td>
<td>“real” priority = criticality + laxity</td>
</tr>
<tr>
<td>Minimum Laxity First</td>
<td>single</td>
<td>all tasks</td>
<td>laxity</td>
</tr>
<tr>
<td>Earliest Deadline First</td>
<td>single</td>
<td>all tasks</td>
<td>deadline</td>
</tr>
<tr>
<td>Rate Monotonic</td>
<td>one per rate</td>
<td>all tasks</td>
<td>rate</td>
</tr>
</tbody>
</table>
Operation Characteristics

- **Criticality** is an application defined significance of the operation missing its deadline.
- **Period** is the time interval between arrivals of dispatch requests for the operation.
- **Execution time** is the longest time used by one execution of the operation.
- **Importance** is a weaker secondary indication of the operation’s significance.

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

- Static priority is assigned by criticality, dynamic priority by laxity, and static subpriority by importance
- Urgency is an ordered tuple with static and dynamic components

Related work:
- Stewart and Khosla '92
- Levine, Gill, Schmidt '98, DASC
- Gill, Levine, Kuhns, and Schmidt '98, submission to IJTCCS
Sources of Schedule Variability

- Phase differences occur when rates are not harmonic
- Variable computation times are produced by operations themselves
- Variable loading is due to variation in system load or to cross-rate dependencies
Limitations of Purely Static Scheduling

- Non-periodic processing is handled inefficiently
- Time cannot be reassigned if an operation is not called, or does not use its worst case computation time
- These limitations can be addressed by pre-allocating sufficient resources to cover worst case behavior
- However, for hardware dependent resources (e.g., CPU time), software may out-pace hardware life-cycles
Limitations of Purely Dynamic Scheduling

- Imposes additional overhead for totally ordering operations
- For under-loaded systems, static approaches may offer comparable average-case behavior
- Canonical dynamic approaches (i.e., EDF and MLF) cannot isolate which operations will meet/miss their deadlines under conditions of overload
A Solution: Hybrid Static/Dynamic Scheduling

- 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

*High Utilization* vs *Isolate Missed Deadlines* vs *Critical/Non-Critical Deadline*
Hybrid Static/Dynamic Scheduling, Cont’d

- **Solution Approach**
  - At configuration time, the scheduling service generates static information using a specific scheduling strategy
    * 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
The Design of TAO’s Strategized Scheduling Service

1. SPECIFY RT_OPERATION CHARACTERISTICS AND DEPENDENCIES

2. POPULATE RT_INFO REPOSITORY

(SCHEDULER’S INPUT INTERFACE)

3. ASSIGN STATIC PRIORITY AND STATIC SUBPRIORITY

4. MAP STATIC PRIORITY, DYNAMIC SUBPRIORITY, AND STATIC SUBPRIORITY INTO DISPATCHING PRIORITY AND DISPATCHING SUBPRIORITY

5. ASSESS SCHEDULABILITY

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)

struct RT_Info {
  wc_exec_time_; //
  period_; //
  criticality_; //
  importance_; //
  dependencies_; //
};
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) addresses additional issues for dynamic and hybrid static/dynamic scheduling.

- This RT-CORBA QoS framework can be extended to support policy and factory driven enforcement of dynamic and hybrid static/dynamic scheduling according to these specifications.

- TAO’s current scheduling approach can be readily adapted to implement this RT-CORBA QoS framework for direct-to-ORB requests.
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 though 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

- **Client-side priority management**
  - RT-CORBA defines a platform independent range of priority values, from `RT_CORBA::minPriority (0)` to `RT_CORBA::maxPriority (32767)`
  - These priorities are mapped into native thread priorities by a default `PriorityMapping`, which an application can replace
  - The client can establish priority-banded connections as well as non-multiplexed connections
Priority management in the RT-CORBA QoS framework

- **Server-side priority management**
  - RT-CORBA provides inbound and outbound server priority transformations
    * Allows server priority inheritance protocols
    * Gives general priority management capability
  - Server priority model determines whether server operates at a fixed priority or adopts the client-side priority of each request
Queue management in the RT-CORBA QoS framework

- **Router priority management**

  - CORBA Messaging defines several router queue ordering policies:
    - ORDER_ANY - the client doesn’t care about the order
    - ORDER_TEMPORAL - requests are processed in the order issued
    - ORDER_PRIORITY - requests are processed in priority order
    - ORDER_DEADLINE - requests are processed in minimum time_to_live order
Queue management in the RT-CORBA QoS framework, Cont’d

module Messaging
{
  // ...

typedef unsigned short Ordering;
const Ordering ORDER_ANY = 0x01;
const Ordering ORDER_TEMPORAL = 0x02;
const Ordering ORDER_PRIORITY = 0x04;
const Ordering ORDER_DEADLINE = 0x08;

// ...

// Router Delivery-ordering Policy
// (default = ORDER_TEMPORAL)

const CORBA::PolicyType
  QUEUE_ORDER_POLICY_TYPE = 35;
interface QueueOrderPolicy : CORBA::Policy
{
  readonly attribute Ordering allowed_orders;
}

// ...
}

- **Router priority management, Cont’d**
  - The priority ordering preserves static priority scheduling
  - The deadline ordering performs EDF scheduling at run-time
  - These policies establish precedents for placing static and dynamic QoS enforcement mechanisms at arbitrary points along a request-response path
Dynamic extensions to the CORBA QoS framework

- **Research Issues**
  - Specifying operation characteristics
    - Static scheduling only needs to propagate *priority*
    - Also need to propagate *run-time characteristics* for dynamic scheduling
  - Providing additional (pluggable) policies for QoS enforcement
  - Enforcing QoS requirements and policies
    - Using *factories* to implement configurations of QoS enforcement mechanisms with the appropriate policies

```c
struct RT_Info {
    Time worstcase_exec_time_;  
    Period period_;  
    Criticality criticality_;  
    Importance importance_;  
};
```
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 policy and factory driven enforcement of hybrid static/dynamic scheduling.
Future Work

• Future scheduling work:
  – Support for distributed scheduling
  – Support for additional mechanisms, policies, and operation characteristics
  – End-to-end priority preservation along a request-response path

• Future performance evaluation work:
  – Investigate effects of priority preservation mechanisms on end-to-end performance
  – Investigate performance of various pluggable network protocols
  – Investigate performance tradeoffs of static, dynamic, and hybrid scheduling strategies and enforcement mechanisms
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 RT Performance Results:
  - www.cs.wustl.edu/~schmidt/RT-OS.ps.gz
  - www.cs.wustl.edu/~schmidt/words-99.ps.gz
For Further Information, Cont’d

- **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/QoS-frameworks.ps.gz