Dynamic Scheduling for Avionics Mission Computing

David L. Levine, Christopher D. Gill, and Douglas C. Schmidt
Washington University, St. Louis
{levine,cdgill,schmidt}@cs.wustl.edu

http://www.cs.wustl.edu/~levine/research/scheduling/DASC-98.ps.gz

03 November 1998

Supported by The Boeing Company and DARPA.
Contents

- Avionics domain requirements
- Static scheduling limitations
- Dynamic scheduling tradeoffs
- TAO’s scheduling service: design and performance
- Concluding remarks
- For further information
Avionics Domain Requirements

- Research Challenges
  - Support both periodic and aperiodic processing
  - Implement efficiently and cleanly
  - Readily support platform (hardware/OS) upgrades
  - Reuse designs and implementations

Harrison, Levine, and Schmidt, OOPSLA ’97

http://www.cs/~schmidt/oopsla.ps.gz
Sources of Static 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 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.
- Goal: higher utilization.
- Hypothesis: with dynamic scheduling techniques we can achieve this goal without undue overhead or instability of the schedule under load.
Requirements for Dynamic Scheduling

- Achieve higher *utilization*
  - Schedule more of the unused time
- Preserve *stability* of the schedule under load
  - Isolate missed deadlines to operations that are not critical to the application
Proposed Metrics to Test the Dynamic Scheduling Hypothesis

- **Laxity**: time from operation completion to deadline (laxity < 0 if deadline missed)
- **Latency**: time from operation invocation (arrival of dispatch request) to operation completion
- **Jitter**: latency variation
- **Total CPU utilization**: efficiency
- **Scheduler/ORB CPU utilization**: overhead
- **Criticality** is an application-specified operation significance.

- **Schedulability** indicates whether there are sufficient resources to perform all registered operations.

- Operations are dispatched in order of *urgency*.

<table>
<thead>
<tr>
<th>Scheduling Algorithm</th>
<th>Criticality Levels</th>
<th>Schedulability Basis</th>
<th>Urgency Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Urgency First</td>
<td>at least two: high and low</td>
<td>critical set</td>
<td>“real” priority = criticality + dynamic priority</td>
</tr>
<tr>
<td>Minimum Laxity First</td>
<td>single</td>
<td>all tasks</td>
<td>dynamic priority</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>unique criticality per rate</td>
<td>all tasks</td>
<td>rate</td>
</tr>
</tbody>
</table>
Maximum Urgency First

- Static priority assigned by criticality
- Dynamic priority assigned by laxity
- Static subpriority assigned by importance

Stewart and Khosla '92
Gill, Levine, and Schmidt '98, submission to IJTCCS
Meeting Request Deadlines

- **Research Challenges**
  - Specifying/enforcing QoS requirements
  - Focus on *Operations upon Objects*
    * Not on threads or communication channels

- **Initial assumptions**
  - Static scheduling
  - Non-distributed

- **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_;  
  Period period_;    
  Criticality criticality_;  
  Importance importance_;  
};
```
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.

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

- **Criticality** is an application-specified operation significance
- **Schedulability** indicates whether there are sufficient resources to perform all registered operations
- Operations are dispatched in order of *urgency*
- Arbitrary mappings from operation characteristics to urgency may be defined and evaluated for effectiveness as scheduling techniques
Real-Time CORBA QoS Support

- **Research Challenges**
  - Specify QoS requirements
    * CPU
    * communication channels, memory, etc
  - Reduce demultiplexing latency
  - Reduce presentation layer overhead
  - Meet deadlines (schedule)
    * static and dynamic, single-CPU
    * dynamic, distributed

- **Related work**
  - Zinky, Bakken, and Schantz, ’95
  - Lee, Yoshida, Mercer, and Rajkumar ’96
TAO’s CPU Scheduling Service

- An RT_Info describes characteristics of one operation
- An RT_Operation wraps an RT_Info and holds the set of all dispatches of that operation
- A Dispatch_Entry describes one dispatch of an operation
Real-time Scheduling Service Use-case

1: I/O SUBSYSTEM receives request from client

2: RUN-TIME SCHEDULER determines priority of request

3: REQUEST QUEUED according to request priority

4: REQUEST DEQUEUED by thread with suitable OS priority

5: REQUEST DISPATCHED to SERVANT

struct RT_Info
{
  wc_exec_time_;  
cached_exec_time_;  
period_;  
importance_;  
dependencies_;  
};

Work Operation

Operation 1: I/O SUBSYSTEM receives request from client

Operation 2: RUN-TIME SCHEDULER determines priority of request

Operation 3: REQUEST QUEUED according to request priority

Operation 4: REQUEST DEQUEUED by thread with suitable OS priority

Operation 5: REQUEST DISPATCHED to SERVANT
The Cost of Dynamic Scheduling

- Solaris 2.5.1/Ultra 30
- Server and 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 Performance on a Commercial RT Platform**

- LynxOS 3.0.0/MCP750
- Server and client on same CPU
- One high priority (20 Hz) client, varying number of low priority (10 Hz) clients
- TAO preserves high priority client response latency and jitter with increasing number of low priority clients

**TAO Two-way Request Latencies**

---

Washington University, St. Louis
Distribution with TAO on Commercial RT Platform

- LynxOS 3.0.0/MCP750/CompactPCI
- Server and client on different CPUs, and communicate via TCP/IP over Fast Ethernet
- High priority client response unaffected by distribution
- Low priority client response improves due to parallelism
- Distribution has small impact on jitter
Concluding Remarks

- Dynamic scheduling supports higher utilization and distribution.
- Dynamic scheduling overhead appears sufficiently small.
- Future scheduling work:
  - Support for distributed scheduling
  - Support for adaptive scheduling
- Future performance evaluation work:
  - Investigate performance on other real-time operation systems
  - Investigate performance on general purpose operation systems with real-time support, including Solaris 2.6, NT 4.0, and Linux 2.0.x w/KURT.
  - Investigate performance tradeoffs of dynamic scheduling.
For Further Information

- **TAO Scheduling:**
  http://www.cs.wustl.edu/~levine/research/scheduling/TAO.ps.gz

- **TAO RT Performance Results:**
  http://www.cs.wustl.edu/~levine/RT-ORB.ps.gz

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

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

- **These slides:**
  http://www.cs.wustl.edu/~levine/research/scheduling/DASC-98.ps.gz