Walther Maldonado Moreira  
*University of Neuchâtel (UNINE), Switzerland*

Pascal Felber  
*UNINE*

Gilles Muller  
*INRIA, France*

Julia Lawall  
*INRIA*

Etienne Rivière  
*UNINE*

Towards Fair Transaction Scheduling with Competing QoS Requirements
Table of Contents

- Context
  - Soft Real Time Applications
- Design
  - Deadline-Aware STM
  - Concurrent Deadlines
- Experimental Results
- Conclusion
Context

Soft Real Time Applications

- Performance is not measured by raw throughput, but perceived responsiveness.
  - Service Level Agreement (SLA)
  - Quality of Service (QoS)
    - Measured as a % of *key events* completed within a given time (*deadline*).
    - e.g.: QoS(95,500) → 95% complete under 500ms
- Missed deadlines do not terminate the application
  - But perceived performance degrades
  - Results in user “frustration”
## Context

### Soft Real Time Applications

- Bulk of user applications falls in this category
  - Multimedia playback (video/audio rendering)
    - Failed deadlines lead frames skipped
    - Leads to choppy video/audio
  - Interactive applications (latency to input)
    - Temporarily “frozen” application
    - Increases user input error rates
  - Gaming (latency to input)
    - High latency handicaps players
    - Has a direct effect on fairness
Context

Soft RT Applications (Examples)

- Swarm
  - OpenGL rendering application [U. of Rochester]
  - One thread renders the scene
  - Others update the world state
  - QoS translates into output *framerate*
## Context

### Soft RT Applications (Examples)

- **Synquake**
  - Benchmark based on Quake [U. of Toronto]
  - Interactive application based on an online game
  - QoS measures action response time (*lag*)
  - Actions have their own scope / data access patterns
### Context

#### Transactional Memory Suitability

- **TM** is a good candidate for these applications
  - Few actual conflicts
  - Locks often cover more objects than needed
  - **✔** TM can provide better scalability
- **✗** However, there are no time guarantees
  - Transactions retry until the block executes successfully
  - Durations are unbounded

![Transaction Scheduling Diagram](image)
## Design
### Our Proposal

- **How can TM be adapted to meet QoS requirements?**
  - **Previous work is on irrevocable transactions**
    - ✔ Prioritizes a transaction → ensures commit
    - ✗ Limits parallelism
    - ✗ Penalizes throughput even when unneeded
  - **Our approach is state based**
    - Use different execution modes with varying scalable/predictable tradeoffs
    - Switch among them according to time to deadline

<table>
<thead>
<tr>
<th>STMs (default optimistic behavior)</th>
<th>STMs (in irrevocable mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Scalable</td>
<td>− Scalable</td>
</tr>
<tr>
<td>− Predictable</td>
<td>+ Predictable</td>
</tr>
</tbody>
</table>
### Design Execution Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimistic (Invisible Reads)</td>
<td>✔ Highest throughput</td>
<td>-</td>
<td>× Conflict detection is often delayed</td>
</tr>
<tr>
<td></td>
<td>× Conflict detection is often delayed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visible Reads</td>
<td>✔ Allows earlier conflict detection</td>
<td>-</td>
<td>× Slower execution</td>
</tr>
<tr>
<td></td>
<td>× Slower execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Irrevocable</td>
<td>✔ Ensures commit</td>
<td>-</td>
<td>× Serializes execution</td>
</tr>
<tr>
<td></td>
<td>× Serializes execution</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Optimistic** (OPT)
- **Visible Reads** (VR)
- **Irrevocable** (IVC)

- **Scalable**
- **Predictable**
Design Switching Modes

- Decision taken at abort/retry
- Estimate required time to complete in other modes:
  - $L = \text{estimated length in Optimistic}$
  - Visible Reads $= 2 \times L$
  - Irrevocable $= 2 \times L$
- Switch times:
  - $ST_{VR} = 6 \times L$
  - $ST_{IVC} = 4 \times L$
## Design
### Time Measuring

- Sample past execution times
  - Use Vitter's Reservoir algorithm
  - Provides representative distribution of durations
- QoS as a percentile + Sorted reservoir → Expected execution time $L$ to use

![Reservoir Diagram]
### Design

#### Concurrent Deadlines

- Meeting QoS can be impossible with simultaneous deadlines
  - Using irrevocable ensures one TX meets its deadline
  - Others are delayed

- Secondary goal to meeting QoS: balancing deadlines
  - Even distribution of missed deadlines → *fairness*
    - No thread performs overall better than others
    - All missed deadlines within a thread fail “equally bad”
### Design

#### Measuring Fairness

- **Measure by Hit-rate**
  - Percentage of deadlines met
  - Stable hit-rate → no thread has an advantage over than others

- **Measure by Overflow**
  - Amount of time by which deadline was missed
  - Stable overflow → missed deadlines are not “worse” in some cases than others

<table>
<thead>
<tr>
<th>Deadline</th>
<th>Overflow Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Measure by Hit-rate**
  - Percentage of deadlines met
  - Stable hit-rate → no thread has an advantage over than others

- **Measure by Overflow**
  - Amount of time by which deadline was missed
  - Stable overflow → missed deadlines are not “worse” in some cases than others
### Design

#### Contention Managers

- **Basic CMs:**
  - Suicide (lowest overhead)
  - Deadline Aware (priority to deadline)

- **Concurrent Deadlines CMs:**
  - Basic (priority to closest to deadline)
  - Fair (priority to lower average hit rate)
  - Cycles (priority to most average overflow time)
  - Compound (if deadline not missed use basic, otherwise cycles)
Design
Irrevocable Queue

- Default IVC implementation is insufficient for concurrent deadlines
- Has no queue, no ordering (spin-lock based)
  - Using a synchronized queue threads enter irrevocable using a “highest priority first” order
- Priority can be determined:
  - Time of deadline (time stamp)
  - Current overflow time average
Experimental Results

Setup

- TinySTM
- Linux Kernel 2.6.34 SMP 64bit
- AMD Opteron Server
  - Four 2.3 GHz quad-core CPUs (16 cores)
  - 8GB RAM
- Swarm
  - Deadline: 30 fps
  - QoS 99%
- SynQuake
  - Deadline specified for the *attack* action
  - QoS 99%
Experimental Results
Single Deadlines (Swarm)

Legend:
- OPT
- VR
- IVC

Success: commit before the deadline (here in VR mode)
Failure: commit after the deadline (here in IVC mode)

OPT only CM: suicide

IVC only CM: suicide

With deadline for 99th percentile CM: deadline aware

With deadline for 99th percentile CM: suicide

#threads
Experimental Results
Concurrent Deadlines (SynQuake)

- IVC Queue - Off
- IVC Queue - Timestamp
- IVC Queue - AvgCycles

CM - basic  CM - compound  CM - cycle  CM - fair  CM - suicide
Experimental Results
Synquake - Overflow distributions

[Graphs showing CDF of overflow cycles for CM suicide, CM basic, CM fair, CM compound, CM cycle, IVC Queue - AvgCycles, IVC Queue - Off, IVC Queue - Timestamp]
Conclusions

- Single deadlines can be met most of the time
  - With results equivalent to irrevocable
    - But with much lower contention
- Concurrent deadlines have widely varying results
  - CM Cycles has the most stable output and good hit-rate
  - Time stamp based queue improves overall results slightly more than overflow time
  - CM Cycles + Time stamp Queue gives overall best result
- There is not necessarily a forced tradeoff between hit-rate and fairness
**Design Measuring Fairness**

- Measured between threads or within threads
  - Between: per-thread results should have low variance
  - Within: overflow times should have low variance
- Selected two approaches to simplify:
  - Hit-Rate variance between threads
  - Overflow variance within threads

![Bar charts showing Hit Rate and Overflow Time](chart.png)
Experimental Results
Synquake - Hit Rate (Threshold 8)
Experimental Results
Synquake - Overflows (Threshold 8)