Chenyang Lu
Department of Computer Science and Engineering
Washington University in St. Louis

Challenges for Real-Time Systems
- Classical real-time scheduling theory relies on accurate knowledge about workload and platform.

New challenges under uncertainties
- Maintain robust real-time properties in face of:
  - Unknown and varying workload
  - System failure
  - System upgrade
- Certification and testing of real-time properties of adaptive systems

Challenge 1: Workload Uncertainties
- Task execution times:
  - Heavily influenced by sensor data or user input
  - Unknown and time-varying
- Disturbances:
  - Aperiodic events
  - Resource contention from subsystems
  - Denial of Service attacks
- E.g., SCADA for power grid management, total ship computing environment

Challenge 3: System Upgrade
- Goal: Portable application across HW/OS platforms
- Some application "work" on multiple platforms

Existing real-time middleware:
- Support functional portability
- Lack QoS portability: must manually reconfigure applications for different platforms to achieve desired real-time properties
- Profile execution times
- Determine/implement allocation and task rate
- Time-consuming and expensive!
Example: nORB Middleware

- CORBA Objects
  - Server
  - Worker thread
  - Conn. thread
- Client
  - Timer thread
  - Priority queues
- Operation Request Lanes

Challenge 4: Certification
- Uncertainties call for adaptive solutions.
- Adaptation can make things worse.
- Adaptive systems are difficult to test and certify

Adaptive QoS Control
- Develop software feedback control in middleware
- Achieve robust real-time properties for many applications
- Apply control theory to design and analyze control algorithms
- Facilitate certification of embedded software

Adaptive QoS Control Middleware
- FCS/nORB: Single server control
- FC-ORB: Distributed systems with end-to-end tasks

Feedback Control Real-Time Scheduling (FCS) Service
- Developers specify
  - Performance specs:
    - CPU utilization = 70%; Deadline miss ratio = 1%.
  - Tunable parameters:
    - Range of task rate: digital control loop, video/data display
    - Quality levels: image quality, filters
    - Admission control
  - FCS guarantees specs by tuning parameters based on online feedbacks:
    - Automatic: No need for hand tuning
    - Transparent from developers
      → Performance Portability!

A Feedback Control Loop
The FC-U Algorithm

- $U_r$: utilization reference
- $K_u$: control parameter
- $R(0)$: initial rate

1. Get utilization $U(k)$ from Utilization Monitor.
2. Utilization Controller:
   \[ B(k+1) = B(k) + K_u(U_r - U(k)) \] /* Integral Controller */
3. Rate Actuator adjusts task rates
   \[ R_i(k+1) = \frac{B(k+1)}{B(0)} R_i(0) \]
4. Inform clients of new task rates.

The Family of FCS Algorithms

- **FC-U** controls utilization
  - Performance spec: $U(k) = U_r$
  - Meet all deadlines if $U_r \leq$ schedulable utilization bound
- **FC-M** controls miss ratio
  - Performance spec: $M(k) = M_s$
  - High utilization
  - Does not require utilization bound to be known a priori
  - Small but non-zero deadline miss ratio: $M(k) > 0$
- **FC-UM** combines FC-U and FC-M
  - Performance specs: $U_r, M_s$
  - Allow higher utilization than FC-U
  - No deadline misses in “nominal” case
  - Performance bounded by FC-M

Control Analysis

- Rigorously designed based on feedback control theory
- Analytic guarantees on
  - Stability
  - Steady state performance
  - Transient state: settling time and overshoot
  - Robustness against variation in execution time
- Do not assume accurate knowledge of execution time


Dynamic Response

- Rigorously designed based on feedback control theory
- Analytic guarantees on
  - Stability
  - Steady state performance
  - Transient state: settling time and overshoot
  - Robustness against variation in execution time
- Do not assume accurate knowledge of execution time

Implementation

- Running on top of COTS Linux
- Deadline Miss Monitor
  - Instrument operation request lanes
  - Time-stamp operation request and response on each lane
- CPU Utilization Monitor
  - Interface with Linux /proc/stat file
  - Count idle time: “Coarse” granularity: jiffy (10 ms)
- Only controls server delay
Offline or Online?

- **Offline**
  - FCS executed in testing phase on a new platform
  - Turned off after entering steady state
  - No run-time overhead
  - Cannot deal with varying workload

- **Online**
  - Run-time overhead (actually small...)
  - Robustness in face of changing execution times

Set-up

- OS: Redhat Linux
- Hardware platform
  - Server A: 1.8GHz Celeron, 512 MB RAM
  - Server B: 1.99GHz Pentium 4, 256 MB RAM
- Same client
- Connected via 100 Mbps LAN

Experiment
1. Overhead
2. Steady execution time (offline case)
3. Varying execution time (on-line case)

Server Overhead

- Overhead: FC-UM > FC-M > FC-U
- FC-UM increases CPU utilization by <1% for a 4s sampling period.

Performance Portability

Steady Execution Time

- Same CPU utilization (and no deadline miss) on different platforms w/o hand-tuning!

<table>
<thead>
<tr>
<th>CPU Utilization (%)</th>
<th>Server A</th>
<th>Server B</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC-U</td>
<td>70%</td>
<td>98.93%</td>
</tr>
<tr>
<td>FC-M</td>
<td>75%</td>
<td></td>
</tr>
<tr>
<td>FC-UM</td>
<td></td>
<td>74.97%</td>
</tr>
</tbody>
</table>

Steady-State CPU Utilization

Server A

- FC-M enforces utilization spec
- FC-U, FC-UM achieves higher utilization

Steady-state Deadline Miss Ratio

Server A

- FC-M enforces miss ratio spec
- FC-U, FC-UM causes no deadline misses

Average Deadline Miss Ratio in Steady State

- $M_s = 1.5\%$
Robust Guarantees
Varying Execution Time

- Same CPU utilization and no deadline miss in steady state despite changes in execution times!

Tolerance to Load Increase

- Surprise: server crashes under FC-M when execution time increases
- FCS/nORB threads run at real-time priority
- Kernel starvation when CPU utilization reaches 100%
- Tolerance margin of load increase
  - FC-U, FC-UM: margin = 1/Us-1
  - Us=70% → Server can tolerate (1/0.7-1)=43% of increase in execution time
  - FC-M: small and "unknown" margin
  - Inappropriate middleware-level service when execution time can increase unexpectedly

Summary of Experimental Results

- FCS algorithms enforces specified CPU utilization or miss ratio in steady state
- Experimental validation of control design and analysis of FCS
- Performance Portability: FCS/nORB achieves the same performance guarantee when
  - platform changes
  - execution time changes (within tolerance margin)
- Overhead acceptable → FCS can be used online

Summary: FCS/nORB

- FCS/nORB supports robust, performance-portable real-time software
- Program application once → runs on multiple platforms with robust performance guarantees!
- FCS/nORB 1.0 release: http://deuce.doc.wustl.edu/FCS_nORB
- Next: FC-ORB
  - Handle end-to-end tasks
  - Fault tolerance

Adaptive QoS Control Middleware

- FCS/nORB: Single server control
- FC-ORB: Distributed systems with end-to-end tasks

End-to-End Task Model

- Periodic task Ti = chain of subtasks (Tij) on different processors
  - All subtasks run at a same rate
  - End-to-end deadline
  - Task rate can be adjusted within a range
    - Trade-off between video quality and rate
    - Higher rate → better video quality & higher CPU utilization

Overall, the lecture covers the robust guarantees of varying execution times, the tolerance to load increase, the summary of experimental results, and the adaptive QoS control middleware. The end-to-end task model is also introduced to demonstrate how tasks can be managed across different processors with varying rates and deadlines.
End-to-End Utilization Control

- CPU utilization
  - Too high → system overload → crash
  - Too low → poor performance (e.g., poor video quality)
  - Utilization + schedulable bound → meet deadlines
- Uncertainties: varying task execution times
  - Adjust task rates to compensate for variations

---

EUCON - Centralized Control Algorithm

- EUCON (End-to-end Utilization CONtrol)
  - Designed based on Model Predictive Control (MPC) theory
  - Invoked periodically to control the utilizations of all processors

---

End-to-End Utilization Control Service

- Implements EUCON (End-to-end Utilization CONtrol)
- Provides functional and performance portability

---

Challenges of End-to-End Utilization Control

- Multi-Input-Multi-Output (MIMO) control
- Utilizations are coupled due to end-to-end tasks
- Rate change affects all processors in the task chain
- Constraints on task rates
- Stability assurance
**Task Migration**

- Fault model: permanent processor failure
- Subtasks have backups on different processors
- Utilization control + fault-tolerance
  - Automatic controller reconfiguration
  - Handle overload caused by task migration

**FC-ORB Implementation**

- Implemented based on FCS/nORB, nORB and ACE
- Specialized for memory constrained Distributed Real-time and Embedded (DRE) systems
- 7017 lines of C++ code
- Controller is implemented as a Dynamic Link Library (DLL) generated by MATLAB

**Experimental Setup**

- 12 tasks (25 subtasks) and 4 Pentium IV processors
- KURT Linux 2.4.22
- Rate Monotonic Scheduling
- Subtasks on Norbert have backups on other processors

**Goal 1: Robust Utilization Control**

1. Norbert fails.
2. Move its tasks to other processors.
3. Reconfigure controller.
4. Control utilization by adjusting task rates.

**Goal 2: Performance Portability**

- Same utilization - portable performance
- Even on different systems with different computing capacity

**Goal 3: Fault Tolerance**

- Norbert fails.
- Move its tasks to other processors.
- Reconfigure controller.
- Control utilization by adjusting task rates.
**Summary: FC-ORB**

1. Robust utilization control, despite
   - unknown or varying execution times
   - external disturbances
2. Performance portability
3. Fault tolerance, in terms of
   - functionality
   - real-time performance

**Conclusion: Adaptive QoS Control**

- Software feedback control: achieve robust real-time properties under uncertainty
- Middleware: provides reusable adaptive QoS control services to many real-time applications
- Control analysis: facilitates certification of embedded software
- Future
  - Advanced control: event-driven, discrete configurations.
  - Coordination of multiple control policies
  - Sophisticated fault tolerance techniques
  - Certification/testing methodologies

**Reading**

- Control of a single server
  - FCS/nORB: Feedback Control Real-Time Scheduling in ORB Middleware, RTAS'03. (required)
- Centralized control of distributed systems
  - FC-ORB: Enhancing the Robustness of Distributed Real-Time Middleware via End-to-End Utilization Control, RTSS'05. (required)
  - EUCON: Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks, RTSS'05, IEEE TPDS.
- Decentralized control of distributed systems
  - DEUCON: Decentralized Utilization Control in Distributed Real-Time Systems, RTSS'05, IEEE TPDS.

**For More Information**

- Papers: [http://www.cs.wustl.edu/~lu](http://www.cs.wustl.edu/~lu)
- Open source middleware: [http://www.cse.wustl.edu/~lu/aqc.htm](http://www.cse.wustl.edu/~lu/aqc.htm)