A Framework accommodating Categorized Multiprocessor Real-time Scheduling in the RTSJ

Jinsan Kwon[†],
Hyeonjoong Cho[†]
and Binoy Ravindran[‡]

[†]Korea University
Embedded Systems and Real-time Computing Lab.

[‡]Virginia Tech
System Software Research Group

Contents

1. Backgrounds

- RTSJ and its implementations provide very good environment for real-time Java applications
  - User-implemented schedulers can be applied to make scheduling decisions
  - However, it does not have any multiprocessor related features

    "How can we make it useful for multiprocessor based schedulers?"

- Scheduling on multiprocessor systems is far more difficult than on single processor systems
  - There is no the Almighty solution for this problem yet

- Several operating systems are capable of multiple schedulers within their kernels

1.1 Categorization of Scheduling Algorithms

- In 2004, Carpenter et al.[1] presented a taxonomy for multiprocessor scheduling algorithms in two-dimensional space
  - The complexity of the priority scheme
    - Static
    - Dynamic, but fixed within a job
    - Fully dynamic
  - The degree of migration allowance
    - No migration
    - Migration only at job boundaries
    - Unrestricted migration

unrestricted

migration degree

3: Unrestricted migration	(1,3)-restricted	(2,3)-restricted	(3,3)-restricted
2: Restricted migration	(1,2)-restricted	(2,2)-restricted	(3,2)-restricted
1: Partitioned	(1,1)-restricted	(2,1)-restricted	(3,1)-restricted
	1: Static	2: Job-level dynamic	3: Fully dynamic

partitioned/static

fully dynamic

priority complexity

[1] Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J. and Baruah, S. 2004. *A categorization of real-time multiprocessor scheduling problems and algorithms*.

1.1 Categorization of Scheduling Algorithms

- The progenitors of all scheduling algorithms[2] (grouped by priority dynamics)
  - Static: RM
  - Job-level dynamic: EDF
  - Fully dynamic: LLF

- The basic schedulers for each category

migration degree ↑			
	Global RM	Global EDF	…
	Restricted migration RM	Restricted migration EDF	Restricted migration LLF
	Partitioned RM	Partitioned EDF	Partitioned LLF

priority complexity

[2] Müller, D. and Werner, M. 2011. Genealogy of Hard Real-Time Preemptive Scheduling Algorithms for Identical Multiprocessors. In Central European Journal of Computer Science, vol.1, no.3, pp. 253-265, September, 2011 DOI=10.2478/s13537-011-0023-z

1.2 Scheduler Frameworks

- Embracing multiple scheduling algorithms
  - Operating system
    - Kernel
      - Linux
      - LITMUS^{RT}
      - ChronOS
    - Kernel/Application
      - RESCH
  - Middleware framework
    - RTSJ implementations
      - JEOPARD

1.3 RTSJ

- The Real-Time Specification for Java
  - Current official spec: 1.0.2
    - Real-time thread scheduling
    - Memory management
    - Resource sharing
    - Asynchronous event handling
  - Recent 'Alpha' release: 1.1 Alpha 6
    - Processor affinity for threads and event handlers
    - Minor updates, improvements and bugfixes

1.3 RTSJ

- Implementation of the RTSJ

  e.g) Java Real-Time System (v2.2u1)

  – A real-time virtual machine
    - Java HotSpot Server VM + @ (RT features)

  – RTSJ class libraries
    - RTSJ 1.0.2 (full)

  – Java class libraries
    - Java SE 5

1.3 RTSJ

- Available RTSJ implementations

Implementation	**RTSJ Ver.**	**Vendor**	**Remarks**
RTSJ RI	1.1a6 / 1.0.2(6)		Last released in JAN 2009
Sun Java RTS	1.0.2	Oracle	
IBM WebSphere RT	1.0.2	IBM	
JEOPARD	1.1 (JSR282)	JEOPARD Consortium	Uses parallel JamaicaVM from aicas GmbH
Perc Pico	JSR302	aonix	J2ME environment
OVM	1.0.1	Purdue University	Under BSD License

2. The CMRF

- The Categorized
  Multiprocessor
  Real-time scheduling-supporting
  middleware Framework
  - Goal: To provide functions to embrace the progenitor algorithms of each category
    - With minimum changes on existing system
  - Environment
    - OS: Linux, kernel 2.6.x
      - Native POSIX Thread Library
      - (PREEMPT_RT or equivalent kernel preemption patches)
    - RTSJ version: 1.0.2 (partial implementation)
      - RealtimeThread
      - Time and timers
      - Scheduler and scheduling parameters

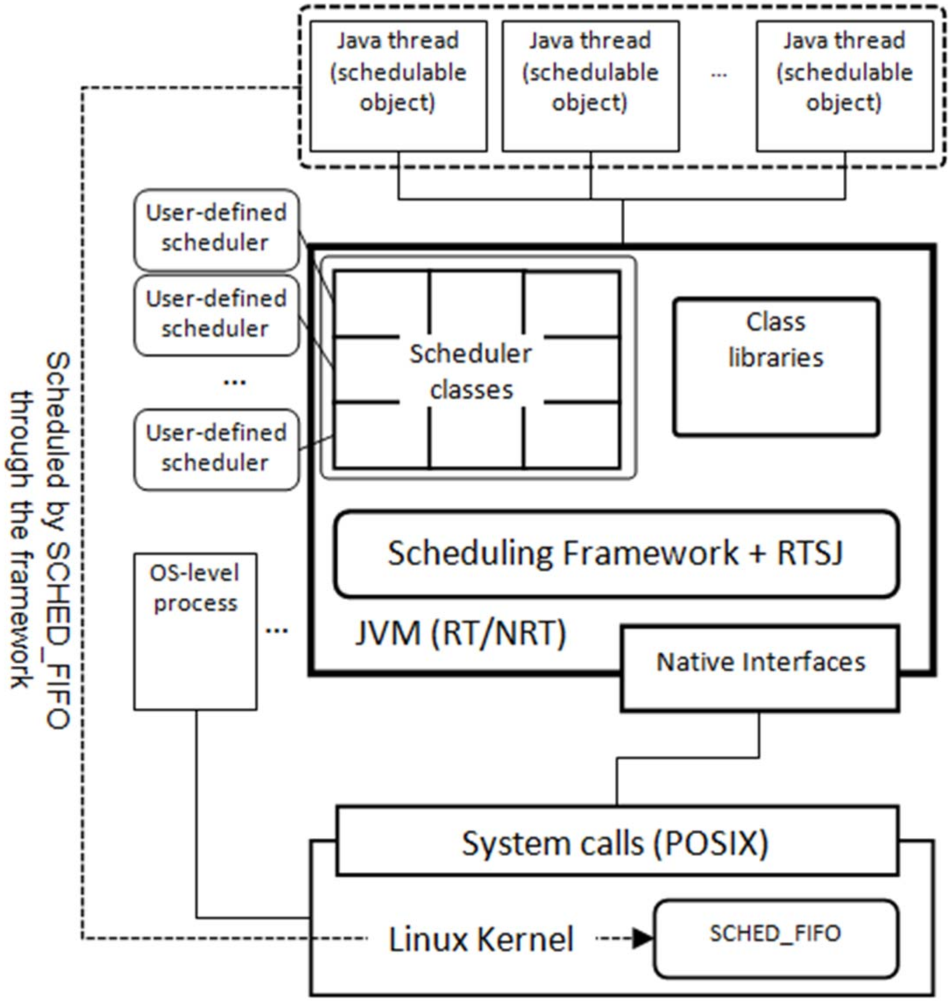
2.1 The CMRF - Structures

- SCHED_FIFO base scheduler
  - Provided by Linux kernel
  - The first-come thread with the highest priority is served first.
  - Serving PriorityScheduler in the RTSJ
- Scheduling 'Schedulables' of the RTSJ
  - Only RealtimeThread is currently supported.
  - Direct use of SCHED_FIFO for thread scheduling
    - sched_setparam: Priority changes
    - sched_setaffinity: Thread migration
- Timers as sleep functions
  - The basic timer is implemented using nanosleep() function provided by underlying OS via Java Native Interface

2.1 The CMRF - Structures

- The structure blocks of the CMRF
  - Native block
  - Java Runtime Environment
  - Framework and RTSJ block
  - Application schedulers

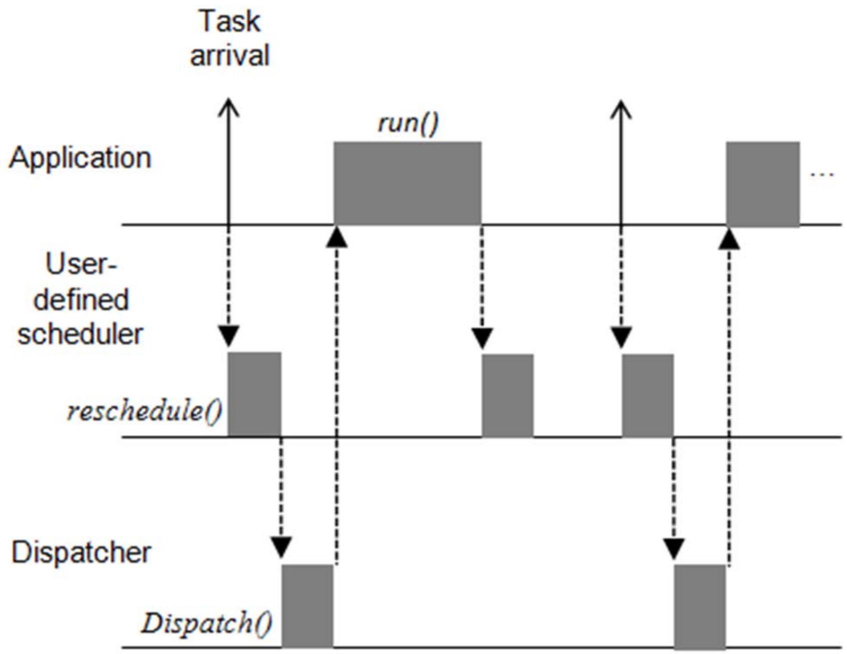
- Scheduling sequence
  - Scheduling decisions are made by application schedulers
  - Within the decision making, the eligible thread is dispatched through the framework
  - The dispatched thread is scheduled by SCHED_FIFO



2.1 The CMRF - Structures

- Scheduling events
  - Arrival of a new RealtimeThread
  - Release of next period of a RealtimeThread
  - Finishing a job
  - Timer

- Thread scheduling flow
  1) Job arrival
  2) reschedule()
  3) Dispatch()
  4) run() method
  5) …
  6) waitForNextPeriod()
  7) (go back to 2)

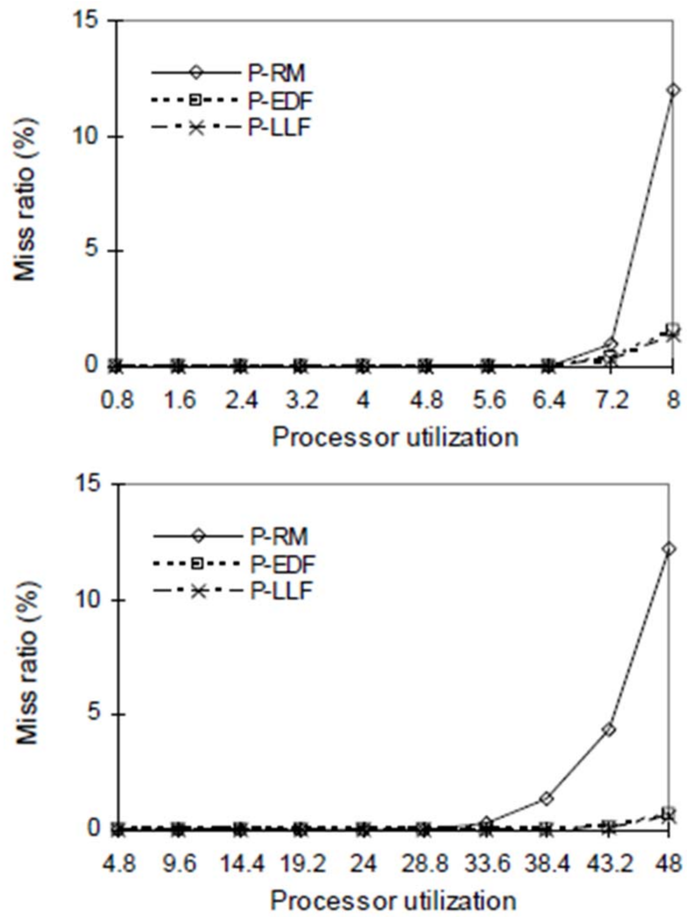


2.2 The CMRF - Evaluation

- Evaluation
  - Task parameters
    - Cost: ≤ 10ms
    - Deadline: 100ms
    - Period: 100ms
    - Total jobs: 100,000 releases / processor
  - Test system
    - Two Intel Xeon E5506 2.13GHz processors, 4 cores per processor
      - 8 total processors
    - Four AMD Opteron 6176 SE 2.3GHz processors, 12 cores per processor
      - 48 total processors
    - Ubuntu Linux 10.04.4, kernel 2.6.31
      - With PREEMPT_RT patch applied
    - OpenJDK6
      - HotSpot Server VM

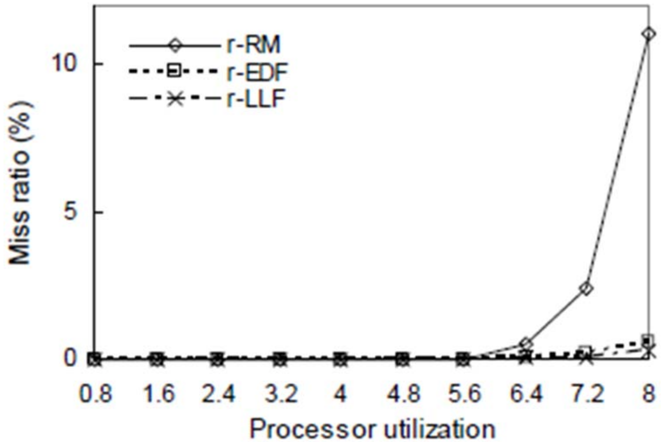
2.2 The CMRF - Evaluation

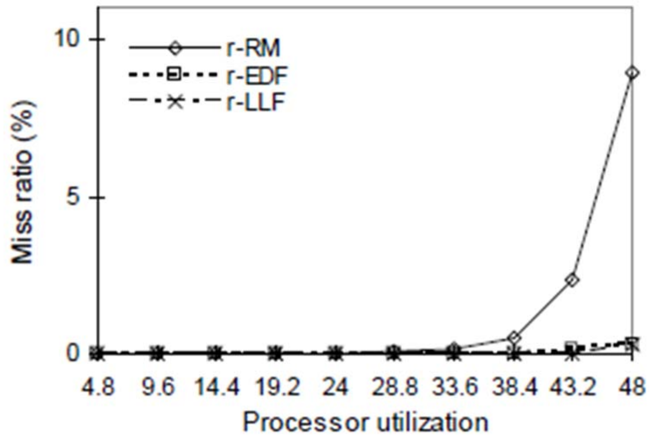
- Deadline-miss ratio test, partitioned
  - RM starts to miss deadlines from utilization of 7.2/33.6 (90%/70%) with miss ratio of 0.95% and 0.31%
  - EDF and LLF starts missing from 7.2/43.2 (90%) with ratio of 0.39%/0.09%(EDF), 0.31%/0.05%(LLF) respectively



2.2 The CMRF - Evaluation

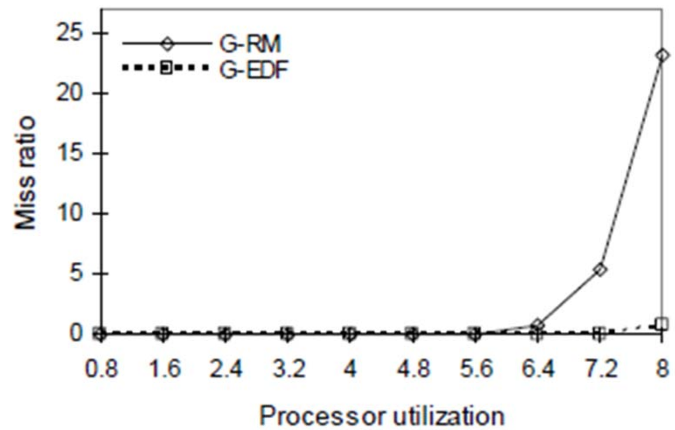
- Deadline-miss ratio test, restricted migration
  - RM starts to miss deadlines from utilization of 6.4 (80%), ratio of 0.5%
  - EDF and LLF starts missing from 6.4/43.2 (80%/90%), 6.4/48(80%/100%) with ratio of about 0.1% and 0.2%

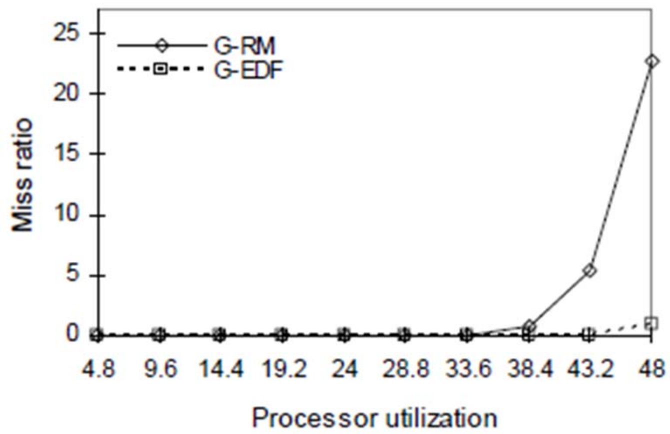




2.2 The CMRF - Evaluation

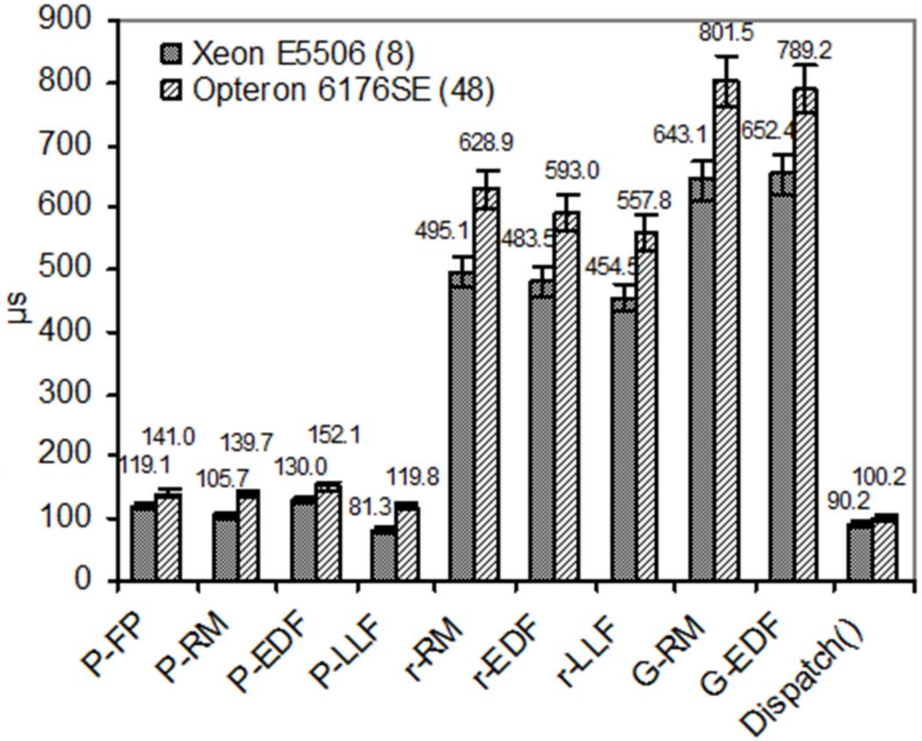
- Deadline-miss ratio test, unrestricted migration
  - RM starts to miss deadlines from utilization of 6.4/38.4 (80%) with ratio of about 0.7%
  - EDF starts missing at 8/48 (100%), ratio of 0.83%/0.98%





2.2 The CMRF - Evaluation

- Elapsed time of each algorithms
  - Measurement of time taken for each algorithm
  - Partitioned schemes takes from 81~152us
  - Dispatch() call takes 90~100us, however, partitioned algorithms do not use affinity assign features in the call
    - As calling Dispatch() frequently, the algorithm gets slower
  - Restricted migration schemes takes around 454~628us
  - Global migration schemes takes 643~801ms



2.2 The CMRF - Evaluation

- Jitter test (within single processor)
  - Measures release overhead
    - Jitter time = [targeted release time – actual release time]
  - Parameters
    - Partitioned
    - Utilization: 0.2
    - Total releases: 160,000 jobs / processor

Intel Xeon E5506			(in nanosec.)
	JRTS	OVM	CMRF
Jitter, Min.	2,438	2,529	2,823
Jitter, Max.	144,426	190,048	52,404
Average jitter	53,546	62,201	52,404
Deviation	20,286	23,017	14,053

AMD Opteron 6176SE			(in nanosec.)
	JRTS	OVM	CMRF
Jitter, Min.	1,457	58	1,308
Jitter, Max.	2,295,608	7,672,276	2,220,614
Average jitter	302,836	417,186	293,807
Deviation	180,192	359,516	117,969

3. Conclusion

- Two-dimensional categorization needs:
  - Priority axis: priority parameters
  - Migration axis: affinity setting
- This can be done by using corresponding system calls and SCHED_FIFO policy
  - Thread scheduling using FIFO policy replaces Schedulables scheduling on the RTSJ
- No other specially built JVM is needed to schedule real-time Java threads on a system

Appendix – Issues and Discussions

- Other major issues
  - Memory management and garbage collection
  - Resource sharing and synchronization

  - High precision & accurate timing (< ms)
    - Current timer is based on nanosleep() call
      - Precise, but not accurate enough to handle (3,3) schedulers
  - Supporting of thread dispatching model:
    - RTSJ 1.1 thread dispatching model by Wellings in [3]
      - More generalized dispatching for non-priority based scheduling

[3] Wellings, A. J. 2008. Multiprocessors and the Real-Time Specification for Java. In *Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing* (ISORC '0 8). IEEE Computer Society, Washington, DC, USA, 255-261. DOI=10.1109/ISORC.2008.22

Thank you for your attention

Jinsan Kwon
mrkwon@korea.ac.kr
http://esrc.korea.ac.kr