

First Look at the Rate Monotonic Analysis

Jaroslav Kačer
jkacer@kiv.zcu.cz

University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

2004-10-18

Presentation Outline

- Current Status of J-Sim
 - Features You May Not Know Yet, Building J-Sim
 - Work in Progress
 - Case Study #1
- Rate Monotonic Analysis – RMA
 - Goals of RMA, History & Background
 - Determining the Number of Priority Levels
 - Schedulability
 - Aperiodic Tasks
 - Task Synchronization
 - Mode Change Protocol

J-Sim Features You May Not Know Yet

- IPC – Semaphores
 - JSimSemaphore, P(), V()
- IPC – Message passing
 - Direct/indirect
 - Blocking/non-blocking
- Random numbers/objects
 - Streams can be initialized → repeatability
- JiJ – simulation of Java multithreading
 - See DSS presentation from 2004-05-17 and Technical Report 2004/03
- Logging

How to Build and Use

- You must have Apache Ant 1.6.2
- build.xml – Description of the whole project
 - Cleans
 - Compiles
 - Generates API documentation, tutorial
 - Generates JARs
 - Library / examples / case studies
- Live show



Work in Progress

- Documentation translation
 - From TeX to DocBook
 - Automatic generation of (X)HTML, PDF, ...
 - PIA
- Putting J-SourceMorph on WWW
 - XML rules for JiJ needed!
- JiJ still not finished completely
 - `join()`, `timed wait()`, `timed join()`
 - Proper unlocking before return / throw

Case Study #1

- Completed as described in the last presentation
- Used JiJ features: locking, wait() / notify(), sleep(), mixing JiJ and classic simulation
- Invariant testing – Several invariants can be temporarily broken
- Live show



Goals of RMA

- Specify, analyze, and predict timing behavior of real-time systems
- Improve dependability of such systems
- Concretely:
 - Assigns priorities to real-time tasks
 - Maximize schedulability of real-time systems
 - Finds whether task deadlines will be met or missed

History

- 1973: CMU – Liu & Layland invent the rate monotonic scheduling theory
- 1980s & 1990s: CMU – RTSIA and RMARTS projects
 - From 1987 to 1993 = 6 or 7 years!
 - Output: ADA 9x, POSIX 1003.4, FutureBus+, ...
- IBM also worked on RMA
- 1990s: Adopted by NASA, ESA, Lockheed Martin, ...

Assigning Priorities

- Why “Rate Monotonic” Analysis?
 - A task's priority depends on its rate
 - The higher the rate (frequency), the higher the priority → Priority is a monotonic function of rate
 - Every task has its period, rate = $1/\text{period}$
- Originally for periodic tasks only
- Priorities are fixed

Determining the Number of Priority Levels (1)

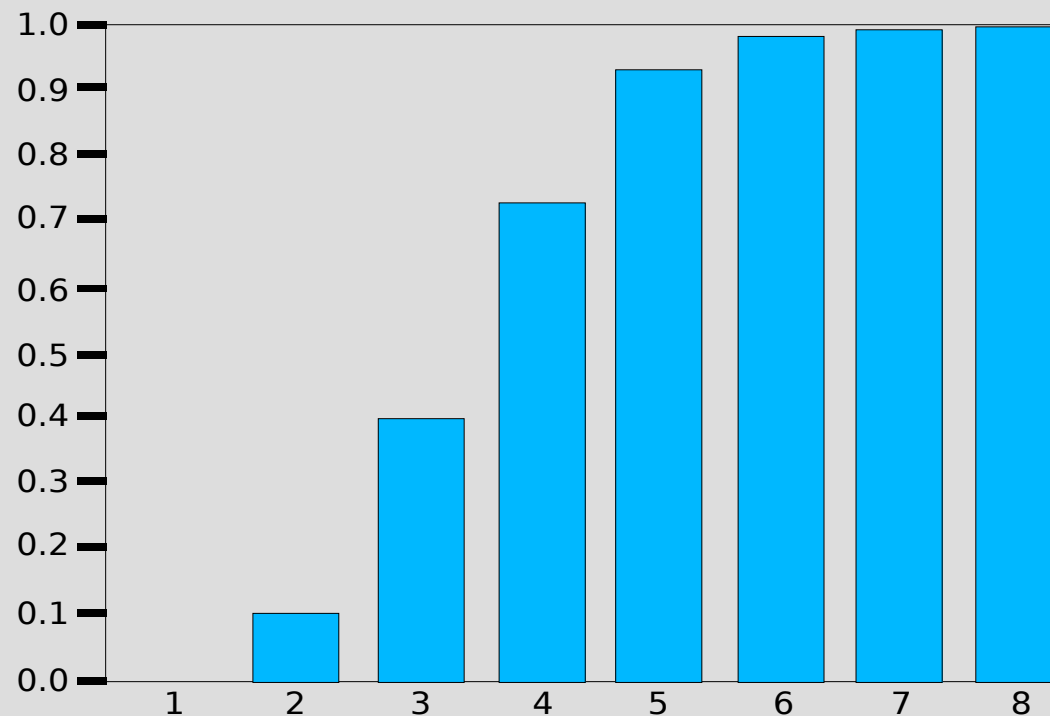
- Ideally: As many as required, not always possible
- Smaller number decreases schedulability
- Constant ratio priority grid:
 - Divide the range $\langle \text{min_period}, \text{max_period} \rangle$ into such a grid that the ratio between two adjacent points is always the same
 - Example: $\text{min} = 1\text{ms}$, $\text{max} = 100\text{ s}$, $r = 2$. Then $L_1 = 1\text{ms}$, $L_2 = 2\text{ms}$, $L_3 = 4\text{ms}$, ..., $L_n = 100\text{s}$

Determining the Number of Priority Levels (2)

- Percentage loss in worstcase schedulability:
loss = $1 - (\ln(2/r) + 1 - 1/r) / \ln 2$
- Example: min = 1 ms, max = 100 s, 256 priority levels
- $r = (L_1/L_0) = (L_2/L_1) = \dots = (L_{256}/L_{255})$
- $r = (L_{256}/L_0)^{1/256} = 1.046$
- Loss = 0.0014

Determining the Number of Priority Levels (3)

- Example: $\text{max} = 100000 * \text{min}$, changeable number of priority bits (and thus priority levels)
- $\text{Schedulability} = f(\text{no. of priority bits})$



Schedulability – Basic Tests

- Periods T_i , Worst-case execution times C_i
- Asymptotic processor utilization bound for n r.m. processes: $U(n) = n(2^{1/n} - 1)$; $\lim = \ln 2$
- Processor utilization for n r.m. processes:
 $C_1/T_1 + \dots + C_n/T_n$
- Schedulability:
 - $\sum(C_i/T_i) < U(n) \rightarrow$ schedulable
 - $U(n) < \sum(C_i/T_i) < 1 \rightarrow$ inconclusive
 - $1 < \sum(C_i/T_i) \rightarrow$ unschedulable

Schedulability – Response Time Test

- If basic test is inconclusive, RTT can be used
- For every task, a_i is computed recursively
- If $a_i <$ task's deadline, task is schedulable
- Compute a_i until a least fixed point is reached
- $a_0 = \sum_{\{j=1..i\}} (C_j)$, $j \dots$ tasks with higher prio
- $a_{n+1} = C_i + \sum_{\{j=1..i-1\}} (\text{ceil}(a_n / T_j) * C_j)$

Aperiodic Tasks

- Aperiodic server
- Execution budget
- Replenishment period
- Budget depleted → Run at background priority until budget replenished
- Types:
 - Priority exchange algorithm
 - Deferrable server algorithm – replenished at every period beginning
 - Sporadic server algorithm – replenished ΔT after spent completely

Task Synchronization (1)

- Danger from synchronization: A low-prio task can block a high-prio task
 - Low, Medium, High prio tasks
 - Low locks the semaphore
 - High preempts, tries to lock, blocked
 - Medium preempts Low
 - Medium blocks High
- Unbounded priority inversion – Duration of priority inversion not bounded by the duration of critical section

Task Synchronization (2)

- Priority ceiling protocol
 - Priority ceiling of $S = \max \text{pr. of } T_i \text{ that can lock } S$
- If task T blockes higher prio tasks, T must execute at the highest prio level of all b. t.
- CS entered only if it executes at a higher prio level than inherited prio level of any preempted CS
- Prio ceilings of all semaphores not hold by T
- Enter CS: If T blocked ($\text{prio} \leq \text{prio ceiling of other } S$), the blocker must inherit T 's priority

Mode Change Protocol (1)

- Deletion or addition of tasks
- Parameter changes of some tasks
- Deadlines must be met before, during, and after
- Tasks:
 - T can be removed if it has not started its execution
 - If T has started, it can be removed after the end of its execution in this period

Mode Change Protocol (2)

- Semaphores:
 - Unlocked S's prio ceiling can be raised immediately
 - Locked S's prio ceiling can be raised after S is unlocked
 - Lower prio ceiling: When all tasks that may lock S and have higher prio are deleted
 - If T's prio affects S's prio ceiling, S's p.c. must be raised before T is added

Possible Pitfalls

- Interrupts
- I/O operations can relinquish CPU time to lower-prio tasks
- Some portions of OS may not be preemptable

Practical Usage

- Influenced Futurebus+, POSIX, ADA 95
- US Navy: submarine sonar system
- ESA: RMA selected as the baseline theory for a hard real-time operating system project
- Lockheed Martin: Avionics applications
- NASA
- IBM

What We Need Now

- How to put RMA and Java together
 - Real-Time Java
- Scheduler or simulator or what?
- RMA is probably much more complex
- Books ordered on RT-Java and RMA