2006

Real-time scheduling algorithms, task visualization

Kevin Churnetski

Real-time scheduling algorithms, task visualization

Kevin Churnetski
Computer Science Department
Rochester Institute of Technology

MS project report, submitted in partial fulfillment of the requirements for the degree
MASTER OF SCIENCE IN COMPUTER SCIENCE

Committee
Professor S. Marshall, Chair
Professor J. Heliotis, Reader
Professor H.P. Bischof, Observer

December 12, 2003

TABLE OF CONTENTS

1 INTRODUCTION 4

2 OVERVIEW OF REAL-TIME SYSTEMS 5

3 DESCRIPTION OF REAL-TIME SCHEDULING 6

4 DESCRIPTION OF SCHEDULING ALGORITHMS IMPLEMENTED 8

4.1 RM SCHEDULING ALGORITHM 8
4.2 EDF SCHEDULING ALGORITHM 8
4.3 LST SCHEDULING ALGORITHM 9
4.4 RR SCHEDULING ALGORITHM 9
4.5 WRR SCHEDULING ALGORITHM 9

5 WORKLOADS AND WORKLOAD GENERATION 10

5.1 WORKLOADS IN THIS PROJECT 10
5.1.1 Workload parameters 10
5.1.2 Generating workloads for experiments 12
5.2 WORKLOAD CHARACTERISTICS PREFERABLE FOR REAL-TIME BENCHMARKS 17

6 RESULTS ON SCHEDULING ALGORITHMS 17

6.1 RM SCHEDULING ALGORITHM, HC0 AND HC1 VS. UTILIZATION 17
6.2 DISTRIBUTION OF SIMULATIONS OVER PARAMETERS AND FAILURE STATISTICS, RM SCHEDULER EXPERIMENTS 20
6.3 COMPARING WRR SCHEDULER TO RM SCHEDULER 24
6.4 COMPARING SCHEDULERS VARYING PREEMPTION AND CONTEXT SWITCH PENALTY 25
6.4.1 Experiments varying preemption and context switch penalty 25
6.4.2 Theoretical optimality of EDF and LST scheduling algorithms 27

7 PUBLISHED RESULTS ON THE RATE MONOTONIC SCHEDULING ALGORITHM 30

7.1 SHOWING CONSISTENCY OF EXPERIMENTS WITH PUBLISHED RESULTS ON THE RM SCHEDULING ALGORITHM 30
7.1.1 RM scheduling algorithm result, asymptotic performance[3] and distribution of periods 30
7.1.2 RM scheduling algorithm result, utilization bounds and harmonic similarity of tasks 32
7.2 DISTRIBUTION OF SIMULATIONS OVER PARAMETERS, RANDOM WORKLOADS WITH UNIFORM PERIOD DISTRIBUTIONS 34
7.3 GENERALIZATION ON PUBLISHED RM SCHEDULING ALGORITHM RESULTS USING PARAMETERS, HC0 AND HC1 35

8 DESCRIPTION OF VISUALIZATION SYSTEM 37

9 CONCLUSION 38

10 REFERENCES 40

APPENDIX A, USER MANUAL FOR REAL-TIME SIMULATOR 41

APPENDIX B, ARCHITECTURAL SPECIFICATION FOR REAL-TIME SIMULATOR 43

APPENDIX C, DATA FILES 49

APPENDIX D, LOG FILE 50

APPENDIX E, DESCRIPTION OF DELIVERABLES 51

TABLE OF FIGURES

Figure 1: *Classification of real-time scheduling algorithms* 7
Figure 2: *Scheduling algorithm acronyms* 8
Figure 3: *Process for generating workloads, method 1 and method 2* 14
Figure 4: *Process for generating workloads, method 3* 15
Figure 5: *Search space partitions for the workloads used to run experiments* 15
Figure 6: *Process for generating workloads with periods uniformly distributed* 16
Figure 7: *Process for simulations and creating charts* 16
Figure 8: *RM scheduler with preemption, 1/HC1 vs. utilization* 18
Figure 9: *RM scheduler with preemption, average 1/HC0 for scheduling failure per utilization* 19
Figure 10: *RM scheduler with preemption, average 1/HC1 for scheduling failure per utilization* 19
Figure 11: *RM scheduler with preemption, simulations per utilization value* 20
Figure 12: *RM scheduler with preemption, percentage of scheduling failures for all values HC0 or HC1* 21
Figure 13: *RM scheduler with preemption, simulations per 1/HC0* 22
Figure 14: *RM scheduler with preemption, percentage of scheduling failures per 1/HC0* 22
Figure 15: *RM scheduler with preemption, simulations per 1/HC1* 23
Figure 16: *RM scheduler with preemption, percentage of scheduling failures per 1/HC1* 23
Figure 17: *WRR scheduler, 1/HC1 vs. utilization* 24
Figure 18: *WRR scheduler, average 1/HC1 for scheduling failure per utilization* 25
Figure 19: *Schedulers with context switch penalty, percentage of scheduling failures per utilization* 26
Figure 20: *Schedulers with context switch penalty, percentage of scheduling failures per context switch penalty* 26
Figure 21: *Theorem 6.4.a example, initially* 28
Figure 22: *Theorem 6.4.a example, first interval shorter* 28
Figure 23: *Theorem 6.4.a example, first interval longer* 28
Figure 24: *Theorem 6.4.b example, initially* 29
Figure 25: *Limiting performance of the RM scheduling algorithm, workload periods uniformly distributed* [3] 30
Figure 26: *RM scheduler with preemption, periods uniformly distributed, percentage of scheduling failures per utilization* 31
Figure 27: *RM scheduler with preemption, periods uniformly distributed, percentage of scheduling failures per 1/HC1* 32
Figure 28: *RM scheduler with preemption, 1/HC1 vs. utilization (range 0.85 to 1)* 33
Figure 29: *RM scheduler with preemption, periods uniformly distributed, simulations per 1/HC1* 34
Figure 30: *RM scheduler with preemption, periods uniformly distributed, simulations per utilization* 35
Figure 31: *Gantt chart* 37
Figure 32: *Screenshot 1 of visualization system* 37
Figure 33: *Screenshot 2 of visualization system* 37
Figure 34: *Directory structure maintained by the user* 41
Figure 35: *Description of contents of directories* 41
Figure 36: *System overview* 43
Figure 37: *Software modules for system* 43
Figure 38: *Description of fields in Task software module* 45
Figure 39: *Inheritance graph for classes* 45
Figure 40: *Object dependencies for classes* 46
Figure 41: *Application interfaces* 46
Figure 42: *Task parameters in data files* 47
Figure 43: *Priorities of tasks in data files* 47
Figure 44: *Table of information on included workload files* 48

ABSTRACT

Real-time systems are computer systems that require responses to events within specified time limits or constraints. Many real-time systems are digital control systems comprised entirely of binary logic or a microprocessor dedicated to one software application that is its own operating system. In recent years, the reliability of general-purpose real-time operating systems (RTOS) consisting of a scheduler and system resource management have improved. In this project, I write a real-time simulator, a workload generator, analysis tools, several test cases, and run and interpret results. My experiments focus on providing evidence to support the claim that for the Rate Monotonic scheduling algorithm (RM), workloads with harmonically non-similar, periodic tasks are more difficult to schedule.

The analysis tool I have developed is a measurement system and real-time simulator that analyzes real-time scheduling strategies. I have also developed a visualization system to display the scheduling decisions of a real-time scheduler. Using the measurement and visualization systems, I investigate scheduling algorithms for real-time schedulers and compare their performance. I run different workloads to test the scheduling algorithms and analyze what types of workload characteristics are preferred for real-time benchmarks.

1 Introduction

In this project, I develop a measurement and visualization system to display the scheduling decisions of a real-time scheduler. I try various scheduling algorithms and compare their relative performance using a visualization system. The effectiveness of the visualization system is evaluated. I include a functional specification of the software tools developed in this project in appendix A (*User manual*) and an *architectural specification* in appendix B. The experimental results are charts created using data from the experiment results.

In addition to testing several algorithms, I test several classes of workloads. Usually, several different algorithmic approaches may successfully schedule any given workload [2]. I test the scheduling algorithms using workloads that push the envelope of schedulability in order to determine which algorithms are the most robust. The research to find these types of workloads provides insight into the characteristics of workloads for use as benchmarks to test real-time schedulers. All workloads in this project have tasks of two types: hard and soft

deadlines. Hard real-time tasks must be completed before a chosen deadline to avoid total system failure.

The problem addressed in this project is the lack of a comprehensive method to find the most difficult to schedule workload for a particular scheduling algorithm that is independent of utilization (which I precisely define). My hypothesis is 1) that there exist workloads W and W' such that utilization of W is less than or equal to the utilization of W' but W is schedulable and W' is not schedulable and 2) the property that makes W' more difficult to schedule can be formulated.

2 Overview of real-time systems

Hard real-time tasks are required to complete computation or execution within the specified timelines. The task must be completed correctly and within the time limits or the system may be considered a total failure. Depending on the specific application, a total system failure could result in catastrophic damages and possibly loss of life. Soft real-time tasks include a time of preferred completion but not a strict deadline. The deadlines of soft real-time tasks may be missed occasionally; in many applications, these are statistical constraints. One example is if the average number of missed deadlines per minute is greater than two then the system is considered to have failed [8].

Typical real-time systems involve a control aspect and a data input aspect [4]. Often, digital controllers and devices provide the interface between the physical environment and more complex real-time computers. These devices may include operator (human) input panels. The interface between a real-time computer's output and the environment may or may not include digital controllers.

Examples of hard real-time systems are the flight controls of an airplane, fuel injection for automobiles, traffic light controls, guidance and propulsion system control of ocean cargo and cruise ships, the mission control computer of the space shuttle, robotic rovers used in mars exploration and medical applications such as computer assisted surgery or life support [8]. There exist many distributed applications with real-time considerations. The issues introduced by real-time, distributed systems are beyond the scope of this project. It is worth noting that Liu [8] among others claim that single processor scheduling techniques can be generalized to have influence on research in multiple processor scheduling.

Examples of soft real-time systems are online database systems and transaction systems. Another example is user interfaces for computer applications and electronic games [8]. Most user interfaces have soft real-time constraints because the human user is interacting with the system directly and prefers timely responses.

Real-Time Operating Systems (RTOS) are available in open-source and commercially in all price ranges. They differ in size (memory space), and in available options/features. Those that are more expensive typically require expensive software tools to complete development of applications. Examples of RTOS are VxWorks, RTLinux, LynxOS, QNX, and Windows CE. Embedded platforms are more frequently the target for RTOS. Relatively few RTOS have been written for the PC (there is a version of QNX that runs on the PC). A recently available open-source RTOS for PC's is KURT [5].

3 Description of real-time scheduling

Real-time scheduling algorithms have been an active topic of research since the late 1960's. Real-time scheduling algorithms work either dynamically or statically. In static scheduling, all tasks are periodic and the periods are known. A periodic task is a task that repeats a request for the processor at a rate equal to its *period*. A static scheduler runs ahead of time on a separate computer than the target system and specifies the order in which all tasks are to be executed. In a correct implementation of the scheduler, all hard real-time deadlines will be met or the algorithm returns no schedule. As you might expect, unless all information about all tasks is known ahead of time, the system may not use a static scheduling algorithm. Because static scheduling algorithms can be run off-line, the run time of the scheduler is generally not an issue. The scheduler is not taking processor time away from the tasks of the application. In practice, static scheduling algorithms have greater complexity.

With dynamic scheduling, both periodic and aperiodic tasks may be run. A dynamic scheduler is part of the system software for the computer and controls when and for how long tasks execute. A dynamic scheduling algorithm is sometimes referred to as a dispatching rule [6]. The dynamic scheduling algorithms can further be divided into static priority dynamic and dynamic priority dynamic [8]. The classification in figure 1 is not exhaustive in depth or detail.

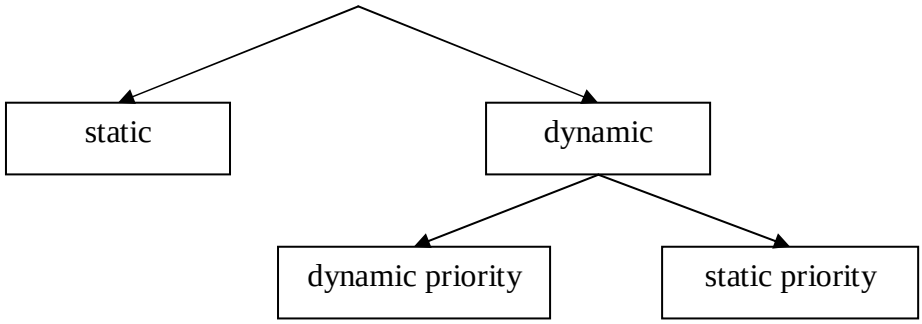


Figure 1: ***Classification of real-time scheduling algorithms***

Static priority dynamic refers to the fact that at any point in the system's schedule, the priority of two tasks in relation to one another is fixed (it is sometimes referred to as fixed priority dynamic). A static priority dynamic scheduler for two tasks A and B that at one point gives priority to A over B must always give priority to A over B. An example of a static priority dynamic scheduling algorithm is the Rate Monotonic algorithm (RM). An RM scheduler prioritizes the tasks by increasing period. The RM is the optimal static priority dynamic real-time scheduling algorithm for periodic task sets. "Any periodic task set of any size will be able to meet all deadlines all of the time if the rate monotonic algorithm is used and the total (processor) utilization is not greater than 0.693" [2]. The application of the RM scheduling algorithm results have proved to be effective when used in combination with other techniques for scheduling periodic and aperiodic task loads as well [3].

A dynamic priority dynamic scheduler for two tasks A and B may at one point give preference to B and later give preference to A. Some examples of dynamic priority dynamic scheduling algorithms are the Earliest Deadline First (EDF) and First-In First-Out (FIFO). In dynamic scheduling practice, techniques that combine several simple heuristic algorithms may also be used.

4 Description of scheduling algorithms implemented

The following scheduling algorithms were implemented for this project. Preemptive and non-preemptive scheduling is allowed with all algorithms except for round robin and weighted round robin, which are only defined for non-preemptive scheduling. A preemptive scheduler has the option to interrupt tasks during their execution. Non-preemptive scheduling requires that a task finish before allowing another task to begin. In my implementation, tasks are released at the beginning of a period and the deadline for a task is at the end of the period. The following acronyms for scheduling algorithms in figure 2 are employed in this report:

Acronym	Meaning
RM	Rate monotonic
EDF	Earliest-deadline first
LST	Least slack time
RR	Round robin
WRR	Weighted round robin

Figure 2: ***Scheduling algorithm acronyms***

4.1 RM scheduling algorithm

The RM scheduler gives higher priority to tasks with smaller periods. An assumption for this project is that the period of the tasks will remain constant. Therefore, whenever a set of tasks are scheduled with the RM scheduler, the tasks are always prioritized in the same order. For this reason, the RM is considered a static priority dynamic scheduling algorithm (as defined previously). As an example, consider three tasks A, B and C with periods 5, 6, and 12 (all execution times are 1). When the tasks are first released, they each request the processor immediately. An RM scheduler first runs task A, then task B and finally, task C.

4.2 EDF scheduling algorithm

The EDF scheduler gives priority to tasks with the closest deadline. While the system executes, the deadlines of each of the tasks will change in relation to the current elapsed time of the system. Although it is true that each of the task's deadlines will change an identical amount of time in relation to each other, EDF is a dynamic priority dynamic scheduling algorithm [8]. The following example illustrates how the priority of two tasks may change using an EDF scheduler: Consider two tasks A and B. Task A has period 3 and execution time 1. Task B has period 5 and execution time 3. The tasks are released at the same time, time equal to 0. Task A has priority over task B and begins execution because its deadline is at time 3 (earlier than B's

deadline of 5). Task A completes at 1. Task B begins execution at 1 and runs for two cycles. At 3, task A is again available for execution but since the deadline of task B is 5 and the deadline of task A is now 6, at this point in time, B has priority over task A. Even if preemption were allowed in this example, the task would not be preempted.

4.3 LST scheduling algorithm

The LST scheduler gives priority to the task with the least amount of slack. Slack is defined as deadline minus remaining work. A task with a large slack number will have a low priority in relation to a task with a small slack number. The task with a small number for a deadline and a large number for a remaining amount of work will have the largest priority. This algorithm can be implemented with a minimum task commitment time to prevent rapid oscillation between tasks when preemption is included. When all tasks in the workload have slacks that differ by less than the minimum task commitment time, the LST scheduler is essentially the same as RR scheduler [8].

4.4 RR scheduling algorithm

The RR scheduler allocates a specified time-slice to each task whose length is known as the quantum. The tasks are in order that is identical to when they arrived into the system. This order is analogous to FIFO. The RR scheduler dispatches tasks in this order. If a task is ready to be dispatched and has no work to be completed, then the next task in the same order is checked. This process continues until a task is found that has work to be completed this cycle or until all tasks are checked. If there are no tasks with work to be done then the system does not do any useful work this cycle. If there is a task with work to be done then the task is dispatched and executed for time equal to the quantum.

4.5 WRR scheduling algorithm

The WRR scheduler is the same as the RR scheduler except that lower priority tasks are executed for a shorter time-slice.

All of the algorithms that I have implemented belong to the dynamic scheduler group. The complexity of the RM, EDF and LST scheduling algorithms using either a heap or balanced binary tree data structure is O(log(n)) for scheduling decisions requiring an insert or delete, where n is the number of tasks. The only tasks that are in the ready queue are those with work to be done before their next deadline. EDF, LST, RR and WRR scheduling algorithms belong to

the dynamic priority dynamic scheduling group. The RM scheduling algorithm belongs to the static priority dynamic scheduling group. The complexity of RR and WRR scheduling algorithms is constant using a FIFO data structure.

5 Workloads and workload generation

This section gives a description of workload generation in this project followed by suggestions on workloads for a real-time operating system benchmark.

5.1 Workloads in this project

First, I give a description of the workload parameters that were employed in this project and aided in workload generation. Then, I describe the workload generation process.

5.1.1 Workload parameters

The following is a description of algorithms for computing the workload parameters.

- **Utilization**: This is a measure by percentage of how much the resources are utilized on the average. In my experiments, the processor is the only resource. Therefore, a workload with utilization = 87% will, on the average, keep the processor running 87 out of 100 seconds (assuming no deadlines are missed) doing the work specified by the tasks of the workload. For a workload of periodic tasks, the utilization is the sum of the utilizations for each task.

$$U(T) = \sum_{i=1}^{n} \frac{(T_i)_X}{(T_i)_P} \qquad (1)\ [2]$$

  where T_i is a task, $(T_i)_P$ and $(T_i)_X$ are the period and execution time of the same task, respectively and n = number of tasks in a workload.

- **Length of simulation**: This value is used in determining the appropriate length of the simulation for a workload in analyzing hard real-time deadlines. I noticed that there are three cases to be concerned about when attempting to find the appropriate simulation time and considering only hard real-time tasks.
  - Case 1: The periods of all tasks are equal (this is actually a special case of the second case). In this case, the simulation repeats each period.
  - Case 2: Every period is a multiple of every smaller period (i.e. periods = 3, 6, and 12). In this case, the simulation repeats after an amount of time equal to the largest period.

- Case 3: Consists of any case besides case 2. In this case, the simulation repeats after a time equal to the least common multiple (LCM) of all the periods (i.e. periods = 3, 9, 12 repeats after 36 cycles).

In all three cases, the LCM will find the maximum simulation time before the simulation repeats. After executing for least common multiple of the periods, the schedule must repeat because all tasks will be in phase equal to the phase that they were in when execution began.

- **Harmonic Coefficient (HC0 or HC1)**: Harmonic similarity separates workloads into three sets as defined in "A Survey of Real-Time Operating Systems – Preliminary Draft" [1]: 1) The workload has periods that are "harmonics of smallest period," 2) uniformly distributed and 3) "in general" (worst case). I have defined two functions to characterize the harmonic similarity of task periods. These functions can be used to separate workloads according to harmonic similarity. By using a function instead of partitioning according to sets, the grouping of workloads becomes more continuous and the separation between them more subtle. I defined two different formulas named harmonic coefficient, HC0 and HC1. The HC1 formula has considerably less range. This made it easier to use in generating workloads. Although the smaller range is obtained with the cost of an approximated value, for the analysis of average results, HC1 seemed to characterize the relationship between periods much better.
  - Harmonic Coefficient 0 (HC0):

$$HC0 = \frac{\max_{i=1}^{n}((T_i)_P)}{LCM_{i=1}^{n}((T_i)_P)} \qquad (2)$$

  - Harmonic Coefficient 1 (HC1):

$$HC1 = \frac{\max_{i=1}^{n}((T_i)_P)}{\max_{i,j=1,i!=j}^{n,n}(LCM((T_i)_P,(T_j)_P)} \qquad (3)$$

The maximum period is included in the HC0 and HC1 equations because it allows the resulting value to be normalized to the lengths of the period. The following is an example: A workload having two tasks such that $(T_1)_P = 2$ and $(T_2)_P = 5$ will have the same HC0 or HC1 as a

second workload having two tasks, such that $(T_1)_P = 4$ and $(T_2)_P = 10$. In the charts, the reciprocal of HC0 or HC1 is used to display the results.

The LCM is needed for calculating the HC0, HC1, and the appropriate length for the simulation. For any two positive integers u and v, LCM(u, v) is the "smallest positive integer that is an integer multiple of both u and v" [11]. This value can be calculated using the following formula:

$$LCM(u,v) = \frac{u \cdot v}{GCD(u,v)} \qquad (4)\ [11]$$

The greatest common divisor of u and v, GCD(u, v) is defined as "the largest integer that evenly divides both u and v" [11] and is calculated efficiently by using the Euclidean algorithm. For HC0, I needed to be able to calculate the LCM of the tasks' periods. The periods of all tasks form a set of positive integers. The Euclidean algorithm is run n-1 times in computing the LCM of this set where n is the number of tasks, noting the following property of LCM for sets of integers:

$$LCM(w,x,y,z) = LCM(w, LCM(x, LCM(y,z))) \qquad (5)$$

For HC1, I needed the maximum LCM of every pair of periods in the workload. Every LCM computation requires the GCD calculation, so by computing the GCD for every pair in the set, the Euclidean algorithm is run $O(n^2)$ times.

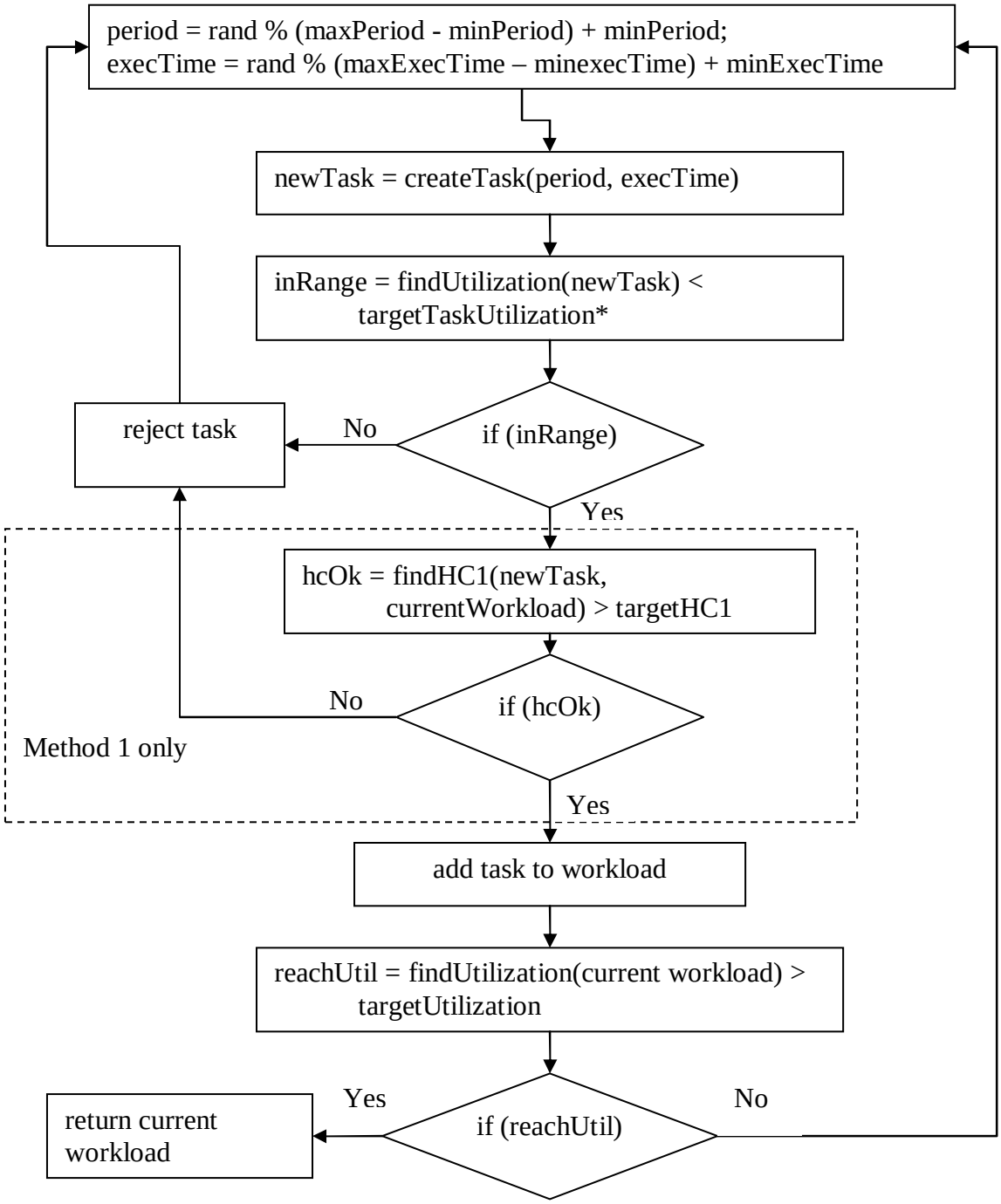
5.1.2 Generating workloads for experiments

An application software module was built to create the workloads employed in experimentation. Random numbers are used to create initial values for the task's execution time and period. The random numbers have a uniform distribution within a range defined by the experiment. After each task is generated, it is added to a workload if none of the workload parameters is exceeded by its introduction. Otherwise, the task is discarded. This process is continued until the workload has parameters that are within a tolerance of specified target workload parameters. Data collected by experiments, for sake of accuracy, contains the actual value for the workload parameters (value that is within tolerance) as well as the results of the simulation.

I found that randomly generated workloads did not generate all combinations of HC1 and utilization. I was unable to find one single, comprehensive method for generating a good distribution of workloads.

In order to fill the entire area of the chart, it was necessary to use three methods (or algorithms) for generating workloads. In the first method, individual workloads are generated according to target experimental parameter indices for HC1 and utilization. By incrementing the target experimental parameter indices, I was able to get an approximately uniform distribution of workloads with respect to both parameters. This method filled the lower two-thirds on a chart varying 1/HC1. The second method generated random workloads according to a target experimental parameter index for utilization only and filled in most of the upper one-third of a chart varying 1/HC1. The periods for the third method's tasks were obtained by randomly choosing prime numbers from an array of two digit primes. These workloads represented the largest 1/HC1 values (top of chart). The first and second methods are shown in figure 3; the third method is shown in figure 4. Figure 5 displays the partitions for the search space of the workloads generated using these three methods.

In order to show consistency with certain published results on scheduling algorithms, experiments were run with workloads that had a uniform distribution over period length. The distribution over HC0, HC1, and utilization was not considered in workload generation for these experiments. The process for generating random workloads with uniform period distributions is shown in figure 6. Statistics about workloads and simulation results from all experiments are included in a later section of this document. As was the case with all strictly random workload generation methods that I tried, there is non-uniformity in HC0, HC1, and utilization distributions of the workloads generated according to a uniform period distribution.



*Unless otherwise noted in experiments, the minPeriod is 10, the maxPeriod is 100, minExecTime = 1, and maxExecTime = 10. The targetNumberTasks = 8 (workloads have approximately 8 to 14 tasks). *targetTaskUtilization = targetUtilization / targetNumberTasks.*

Figure 3: ***Process for generating workloads, method 1 and method 2***

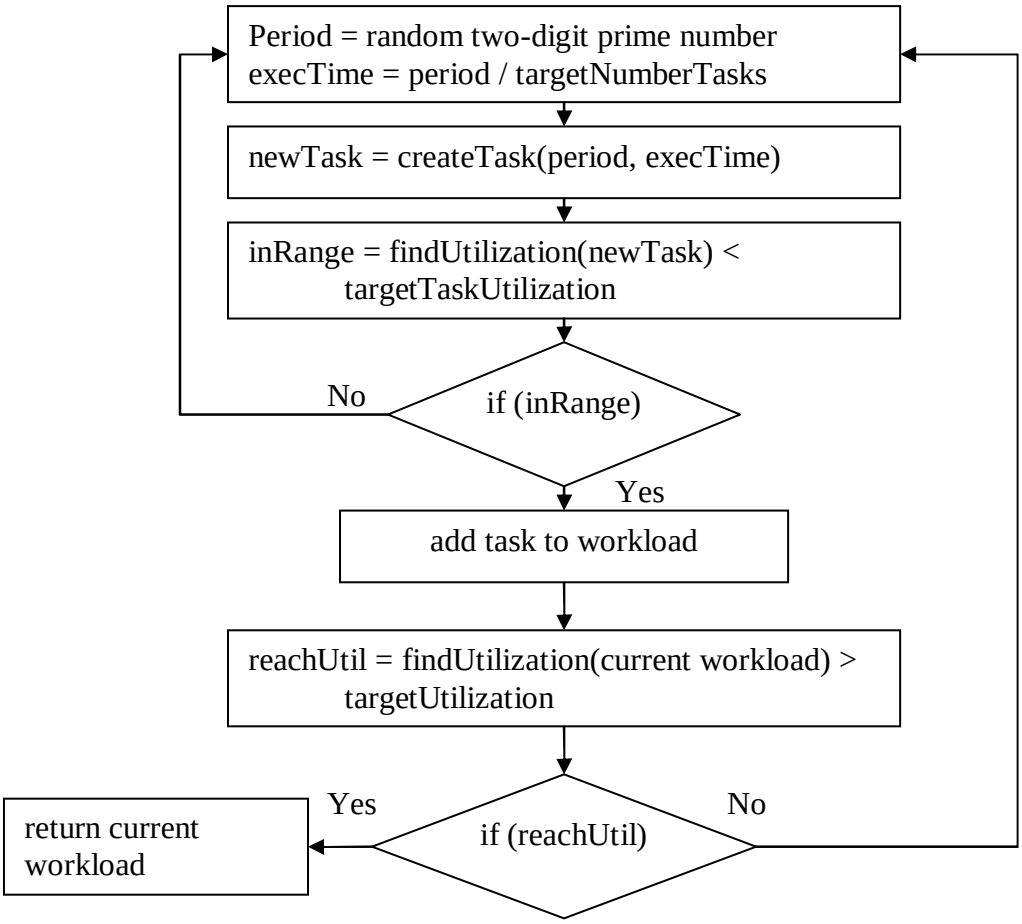


Figure 4: ***Process for generating workloads, method 3***

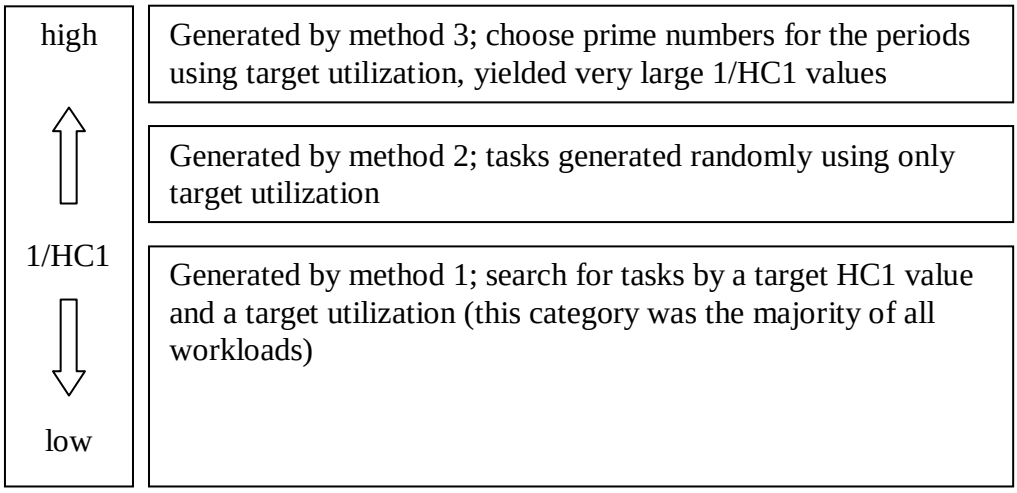
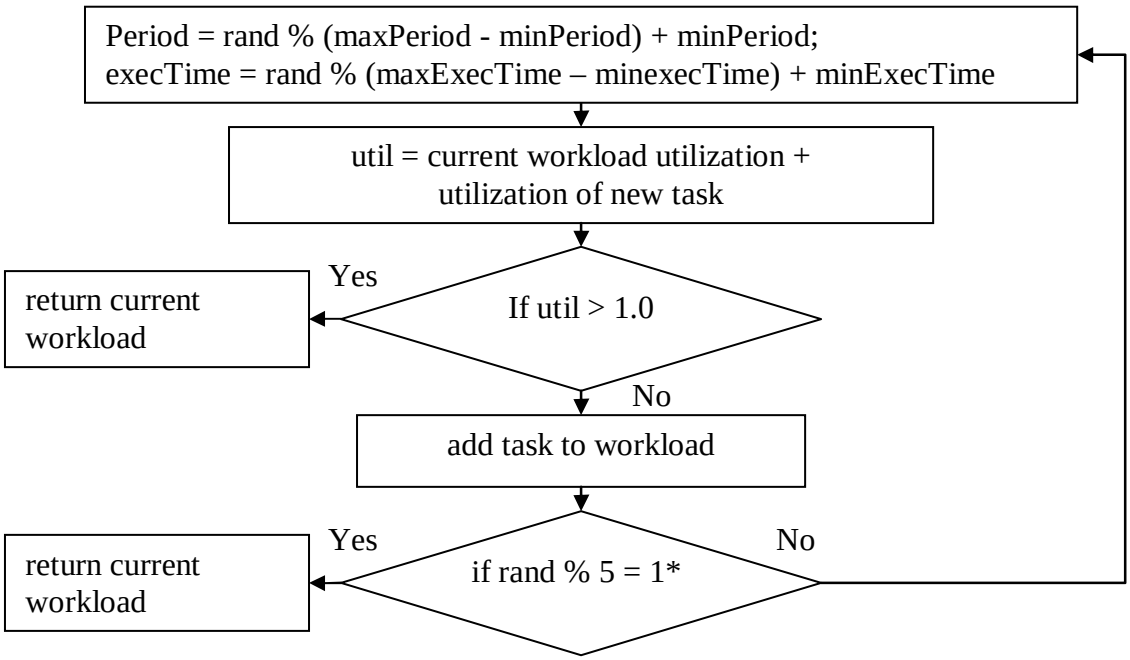
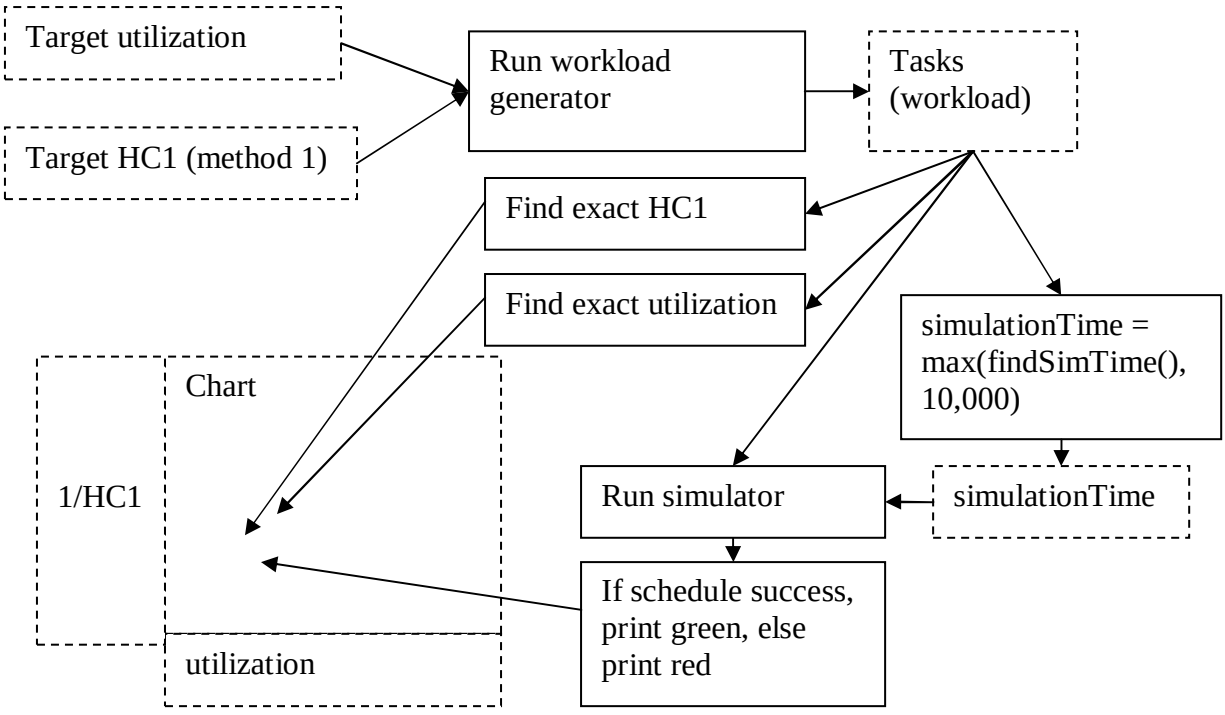


Figure 5: ***Search space partitions for the workloads used to run experiments***



**This step gives workload generation a 1 in 5 chance to complete execution after creating this task (allowing workloads with lower utilization to be in the experiment)*

Figure 6: ***Process for generating workloads with periods uniformly distributed***



Dashed lines represent data elements and the solid lines represent process elements

Figure 7: ***Process for simulations and creating charts***

Figure 7 shows the process followed in running the experiments. The simulation time is bounded by 10,000 simulated seconds. Most schedules that failed were observed to fail in less than this amount of time. The number of simulations for one algorithm on a chart is approximately 15,000. Each data point represents one simulation unless otherwise noted.

5.2 Workload characteristics preferable for real-time benchmarks

For a non-real-time operating system benchmark, the method of judging performance is turn-around time or some other speed of execution characteristic. In real-time systems, the timeliness of task completion is not as important as the guarantee that all tasks will meet their deadlines. Therefore, a real-time operating system benchmark gradually increases scheduling difficulty until a deadline is missed. This can be done either on several experiments as in this project or as one workload that increases scheduling difficulty by introducing new tasks or modifying existing ones. In the interest of accuracy, the more control over scheduling difficulty that the simulation parameters allow, the better. The resulting performance for the algorithm on the benchmark is (a numerical representation of) the most difficult to schedule workload that the algorithm successfully scheduled.

6 Results on scheduling algorithms

The first sub-section gives experiment results and average case results for the RM scheduler. It also describes how harmonic similarity of task periods as characterized by the HC1 and HC0 metrics affects schedulability for the RM scheduler and is independent of utilization. Then, I show the distributions of workloads and percentages of success and failure over parameters for RM scheduler experiments. The distributions of experiments determine confidence in results. The success and failure percentages show the same results although not as conclusively as the previous sub-section. Next, I apply the experimental approach used for the RM scheduler to the WRR scheduler. Finally, I show a comparison of all scheduling algorithms implemented in a setting that varies the context switch penalty.

6.1 RM scheduling algorithm, HC0 and HC1 vs. utilization

Figure 8 shows the effect that HC1 and utilization have on scheduling. A green data point on the chart indicates that the workload was scheduled successfully using the experiment method described previously. A red data point indicates failure to schedule the workload (i.e. a deadline was missed). There is a distorted boundary between the successes and failures

noticeable from the top, center of the chart (1/HC1 = 99, utilization = 0.825). The line appears to be vertical, initially. Moving down and to the right, the slope slowly becomes apparent as utilization increases to 0.85. Continuing down and to the right, the curve of the boundary changes more rapidly (between utilization 0.85 and 0.95). The boundary is almost flat for utilization greater than 0.95 because there is no failures in the 1/HC1 = 1 row (bottom row of chart) and there is many failures directly above.

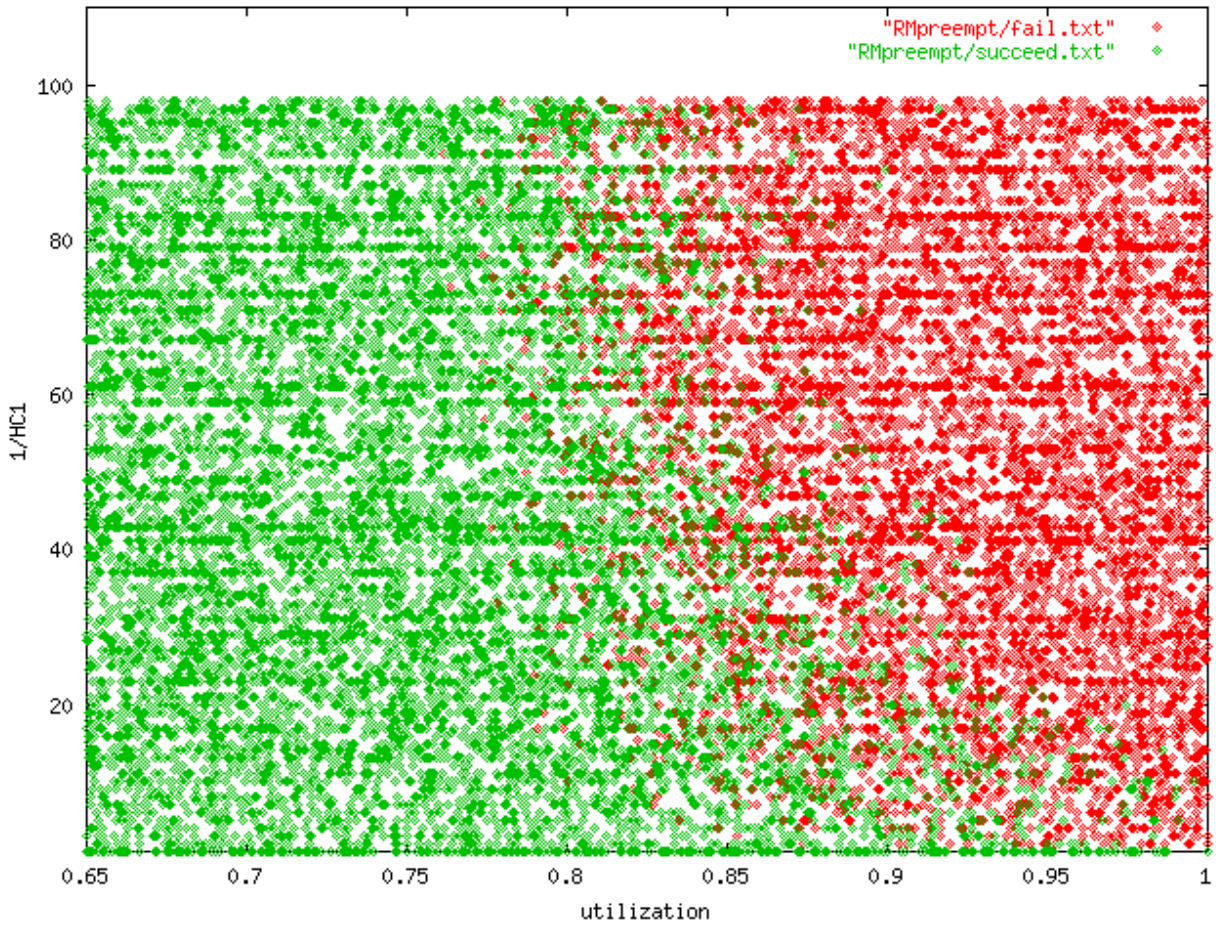


Figure 8: ***RM scheduler with preemption, 1/HC1 vs. utilization***

The slope of the boundary between success and failure shown in figure 8 indicates that although utilization is a prime factor in determining the schedulability of a workload, when 1/HC1 is between 1 and 50, this parameter will also affect schedulability. When 1/HC1 is less than five, the utilization parameter has less effect on schedulability and when 1/HC1 equals one, the utilization has no effect on schedulability because the workloads are always schedulable.

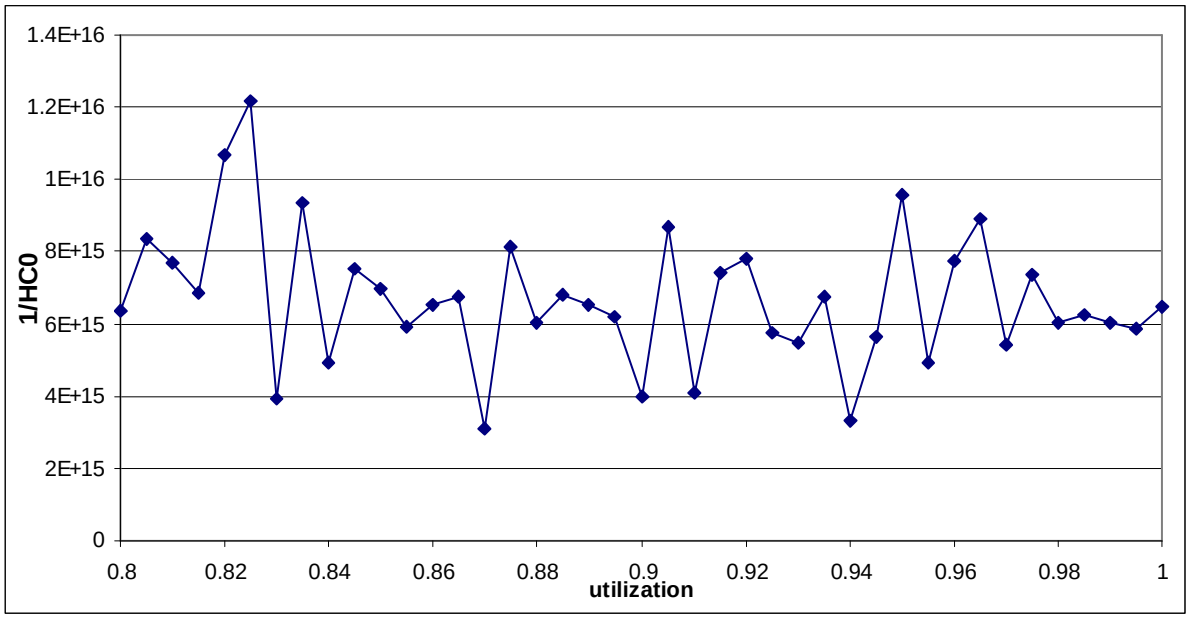


Figure 9: ***RM scheduler with preemption, average 1/HC0 for scheduling failure per utilization***

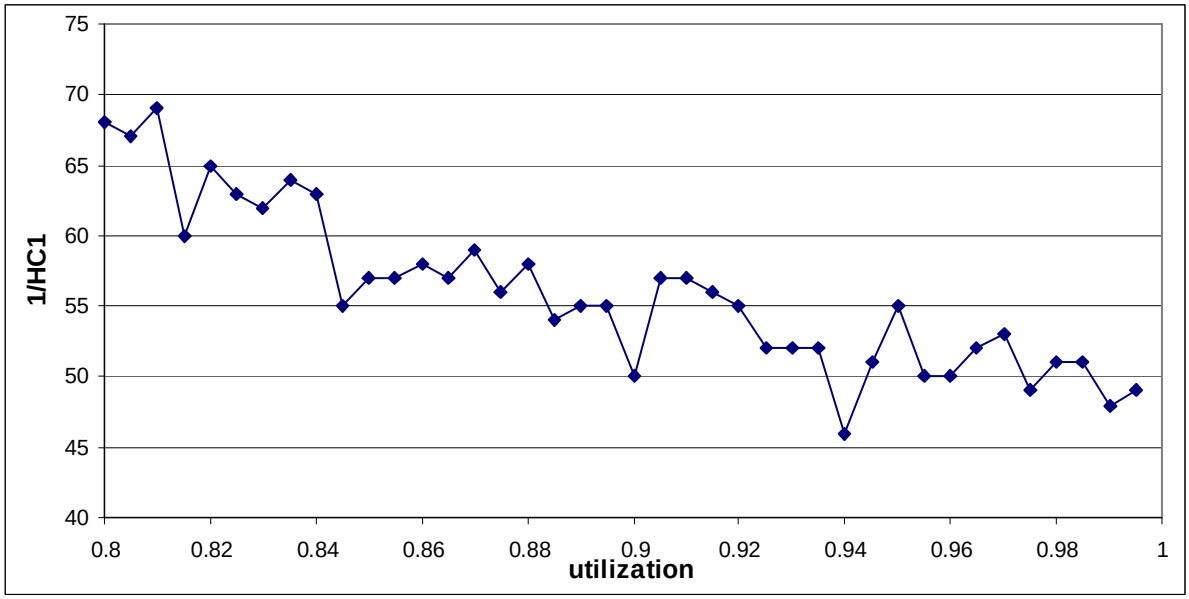


Figure 10: ***RM scheduler with preemption, average 1/HC1 for scheduling failure per utilization***

Figure 9 shows average 1/HC0 for all tasks that failed to be scheduled for a given utilization. Figure 10 shows average 1/HC1 for all tasks that failed to be scheduled for a given utilization. In both charts, the RM scheduler is displayed with preemption. The 1/HC0 and 1/HC1 values for the workloads that failed were summed per utilization then divided by total number of failures. Each data point represents the average 1/HC1 or 1/HC0 value for failure out of approximately 250 experiments (over 10,000 experiments total). Both harmonic coefficient

values were collected on the same experiments. The data for HC1, in the second chart (figure 10) shows a well-defined trend. For increasing utilization, there is a clear downward trend in average 1/HC1 for failure. This supports the claim that the HC1 formula is a good representation of how the harmonic similarity of periods affects schedulability in the presence of utilization increase. The first chart shows that there is no apparent trend in average 1/HC0 for failure (figure 9).

6.2 Distribution of simulations over parameters and failure statistics, RM scheduler experiments

The charts in figure 11 and figure 12 show statistics about the number of workloads per utilization in RM scheduler experiments. Although the workloads were generated using different methods, figure 11 shows that there is a reasonably uniform distribution for total simulations over all values for utilization.

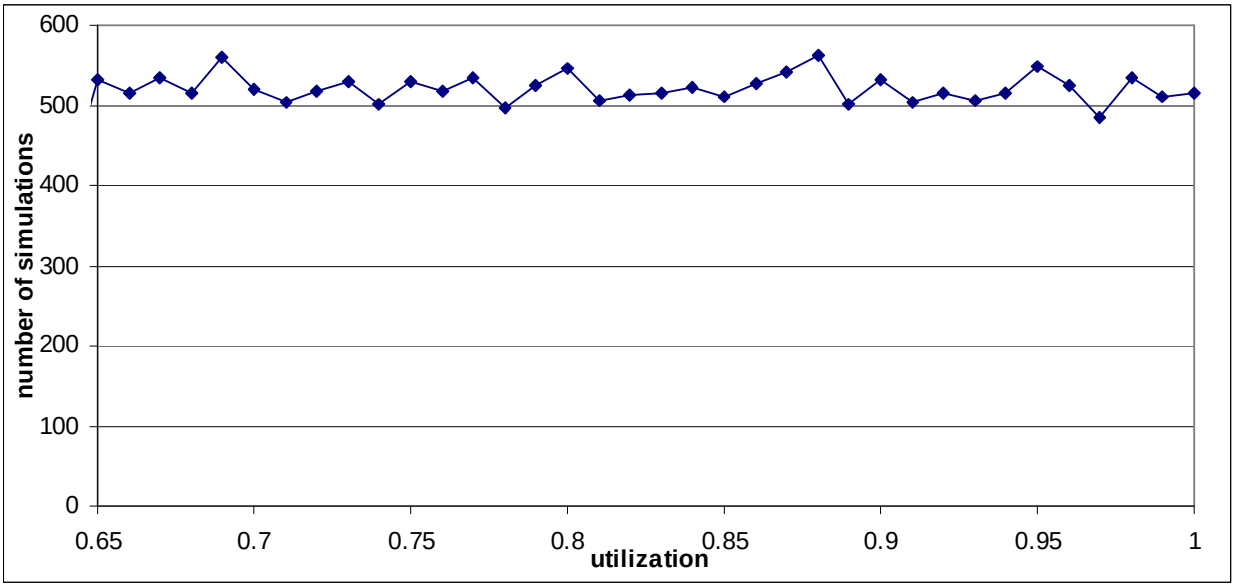


Figure 11: ***RM scheduler with preemption, simulations per utilization value***

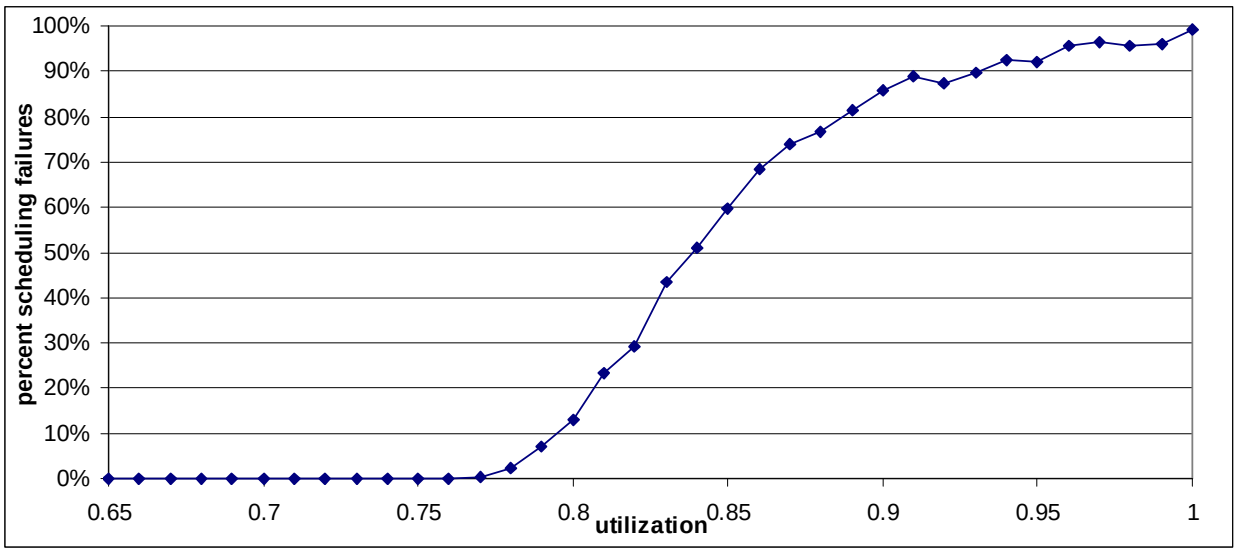


Figure 12: ***RM scheduler with preemption, percentage of scheduling failures for all values HC0 or HC1***

Figure 12 shows the percentage of failures for the RM scheduler with preemption. Unsurprisingly, the data shows that as utilization increases, the number of failures also increases. There is not a significant population of failures, however, until the utilization increases to 0.8. For this reason, in figures 9 and 10 (previous sub-section), the data points with utilization less than 0.8 were removed from the chart because there were not enough simulation failures to draw conclusions from the average.

Figures 13 and 15 show the number of workloads per 1/HC0 and 1/HC1, respectively. Figures 14 and 16 show the percentage of failures per 1/HC0 and 1/HC1, respectively. The distribution of workloads over 1/HC0 rarely falls below 100 for any of the data points (figure 13), but does not have as uniform of a distribution as 1/HC1 (figure 15). The reason workloads have a uniform distribution over HC1 is clearly because HC1 was one of the target parameters for the workload generation routine. It is interesting to note that HC0 (figure 14) characterizes scheduling difficulty with a more direct relationship than the HC1 (figure 16) when considering only percentage of failures. However, since the utilization distribution for each incremental HC0 or HC1 value is somewhat uncontrolled (for both figures 14 and 16), the confidence of these results is not as good as the previous findings.

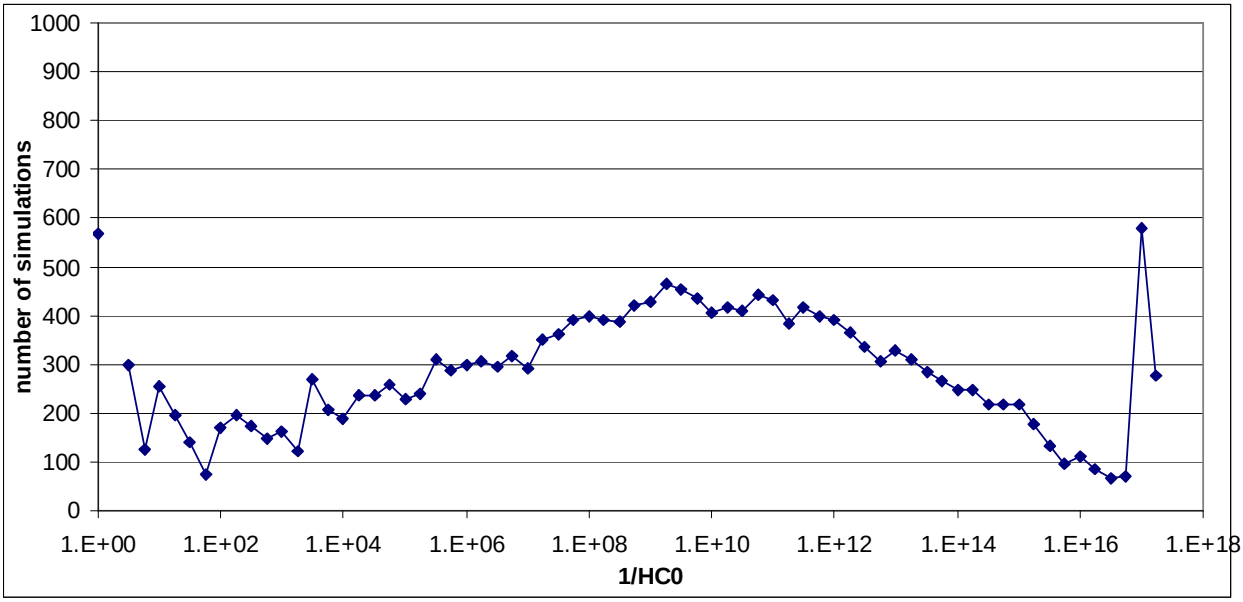


Figure 13: ***RM scheduler with preemption, simulations per 1/HC0***

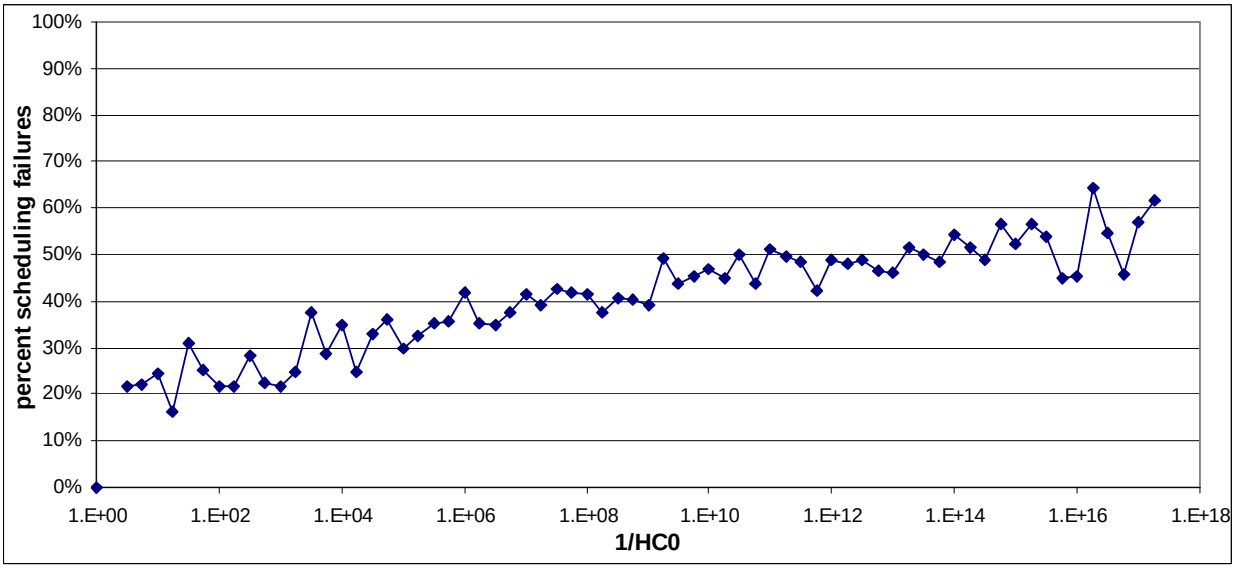


Figure 14: ***RM scheduler with preemption, percentage of scheduling failures per 1/HC0***

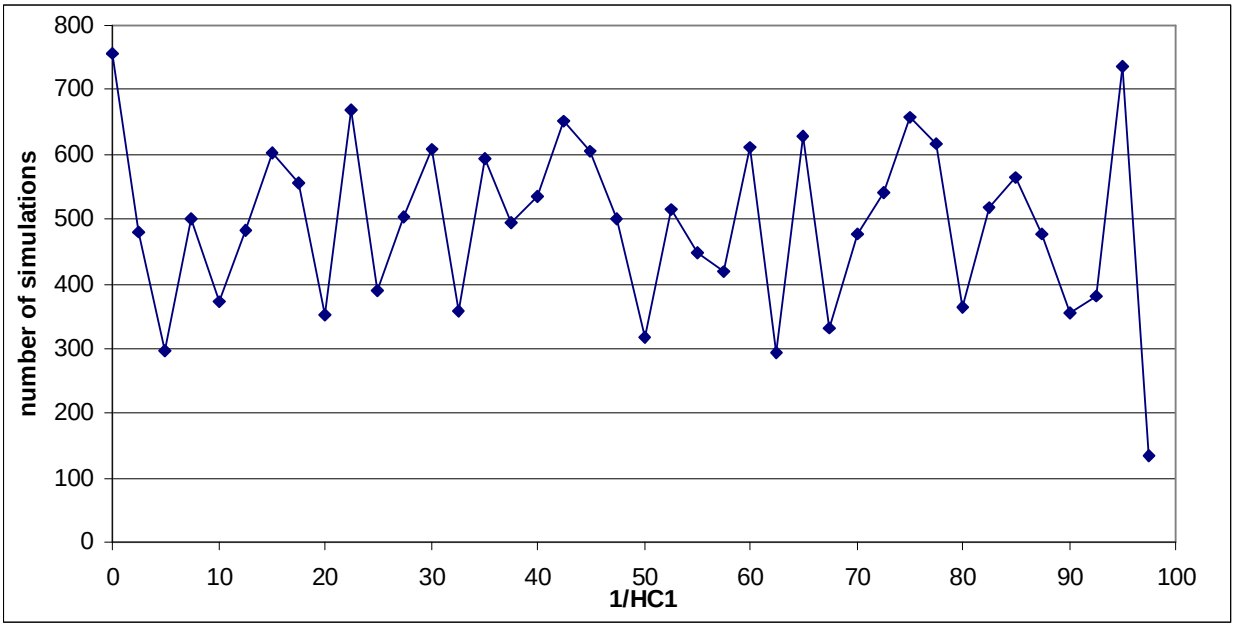


Figure 15: ***RM scheduler with preemption, simulations per 1/HC1***

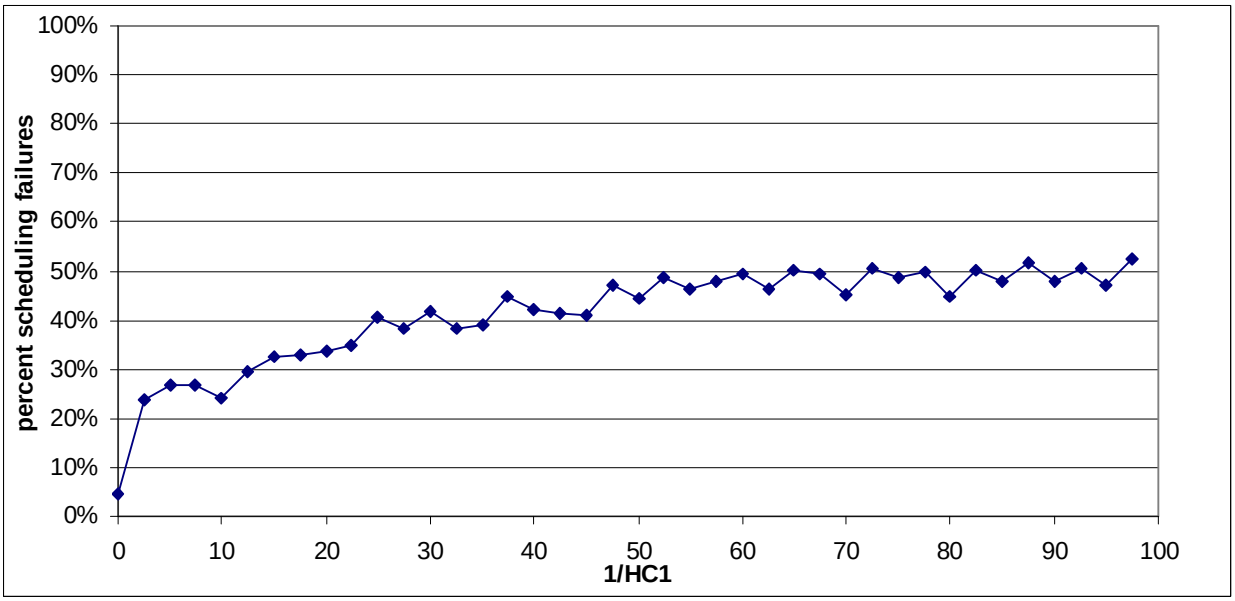


Figure 16: ***RM scheduler with preemption, percentage of scheduling failures per 1/HC1***

6.3 Comparing WRR scheduler to RM scheduler

Figure 17 shows scheduling successes and failures for the WRR scheduler. A success is plotted as a green data point and a failure as a red data point. The most notable differences between the RM scheduler (figure 8, previous section) and WRR scheduler (figure 17) are that the WRR has an increased number of failures and the failures occur with less utilization. The WRR scheduler also has a significant number of failures in the 1/HC1 = 1 row (bottom row of chart) whereas the RM scheduler never fails in this range. Also noticeable is the observation that the WRR scheduler performs better with increasing 1/HC1. This is opposite of the effect that 1/HC1 has on the RM scheduler. It is important to state, however, that the WRR scheduler's performance does not surpass the RM scheduler in any region of the chart. The quantum used for the WRR scheduler experiments is three simulated seconds (all tasks are the same priority, namely, hard real-time).

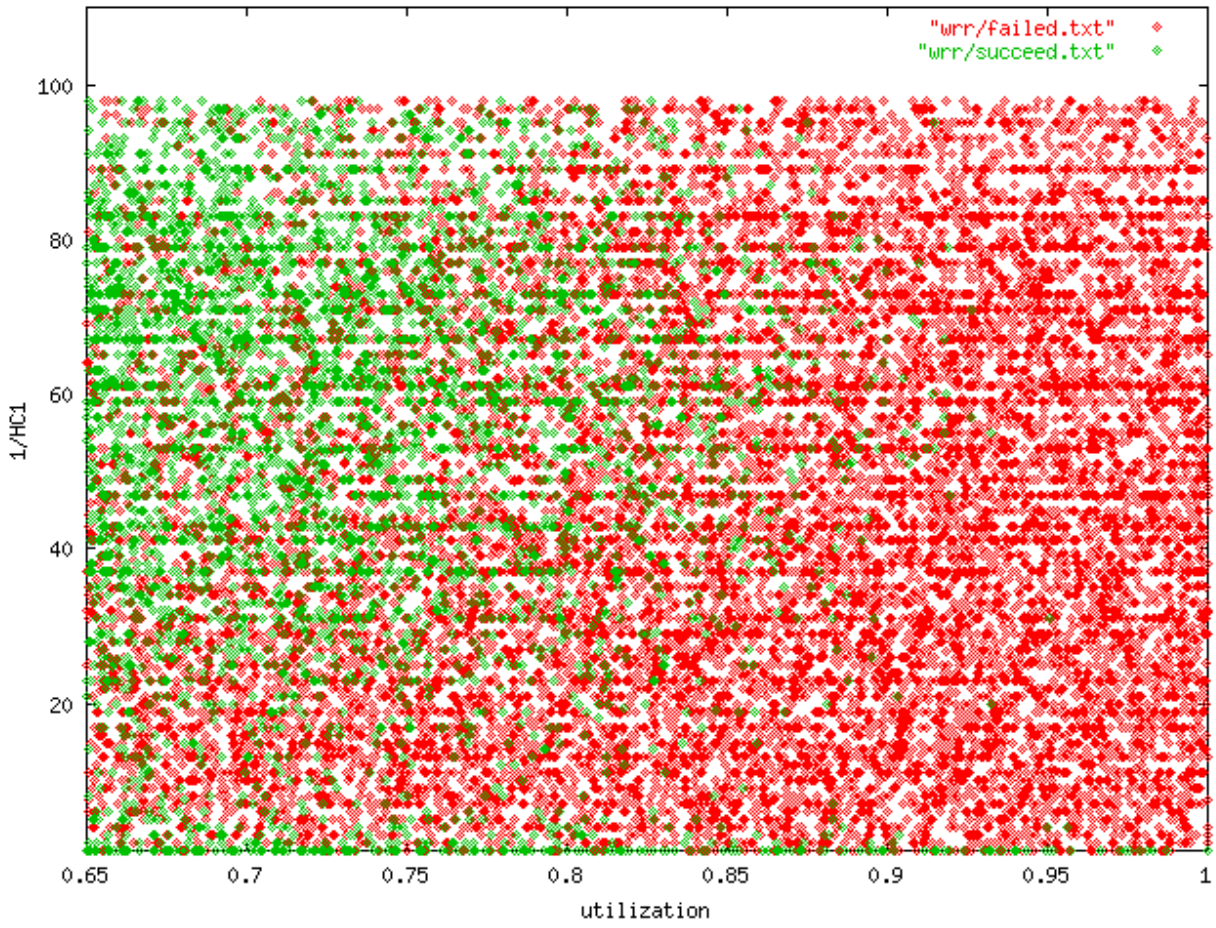


Figure 17: ***WRR scheduler, 1/HC1 vs. utilization***

Figure 18 shows the average 1/HC1 of workloads that fail as a function of utilization. The results show a positive slope contrasting the negative slope for the RM scheduler's corresponding chart (figure 10, previous section).

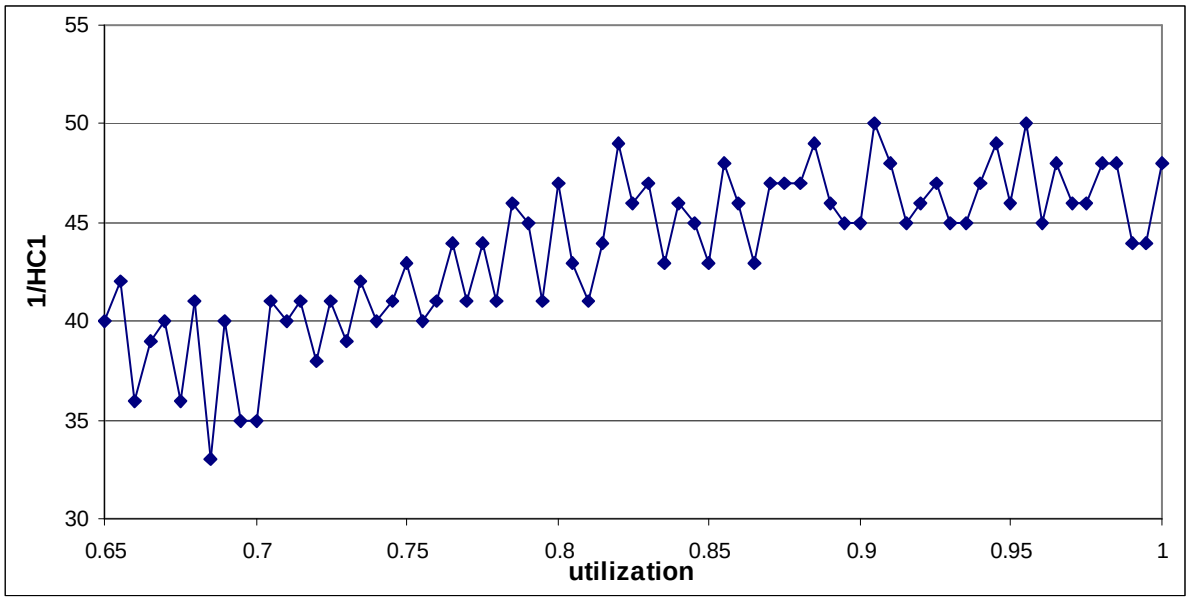


Figure 18: ***WRR scheduler, average 1/HC1 for scheduling failure per utilization***

6.4 Comparing schedulers varying preemption and context switch penalty

The first sub-section compares several scheduling algorithms by looking at their performance in experiments that vary context switch penalties. The second sub-section explains the theoretical optimality of EDF and LST schedulers using preemption.

6.4.1 Experiments varying preemption and context switch penalty

The amount of time required for a processor to complete a context switch typically ranges from 1 to 1000 microseconds [9]. In these experiments, the time units are called seconds, but actually are arbitrary since the task lengths are not analyzed except in relation to one another. If the tasks are assumed to have execution times ranging from 1 to 10 simulated seconds, then the context switch penalties range from 0 to 0.95 seconds using 20 increments of 0.05 seconds. With this same assumption, the minimum task commitment time for the LST scheduler with preemption is 1 second, in these experiments.

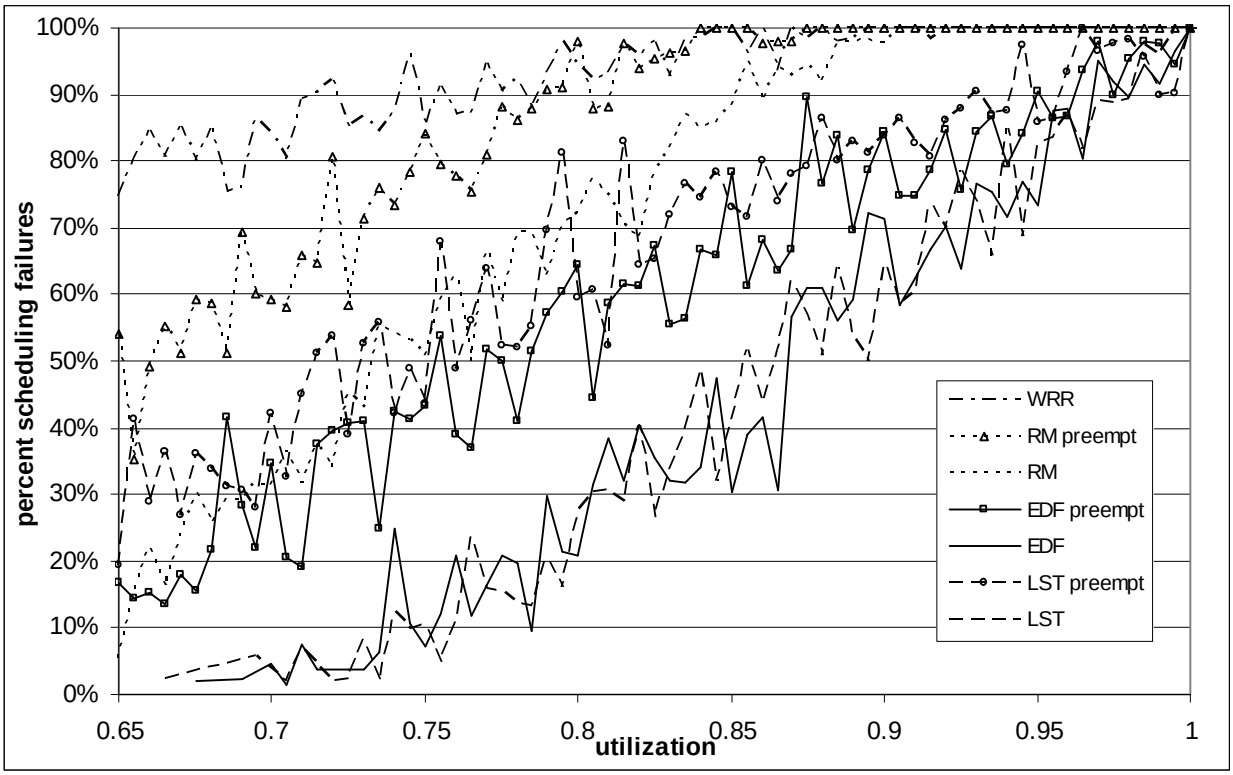


Figure 19: ***Schedulers with context switch penalty, percentage of scheduling failures per utilization***

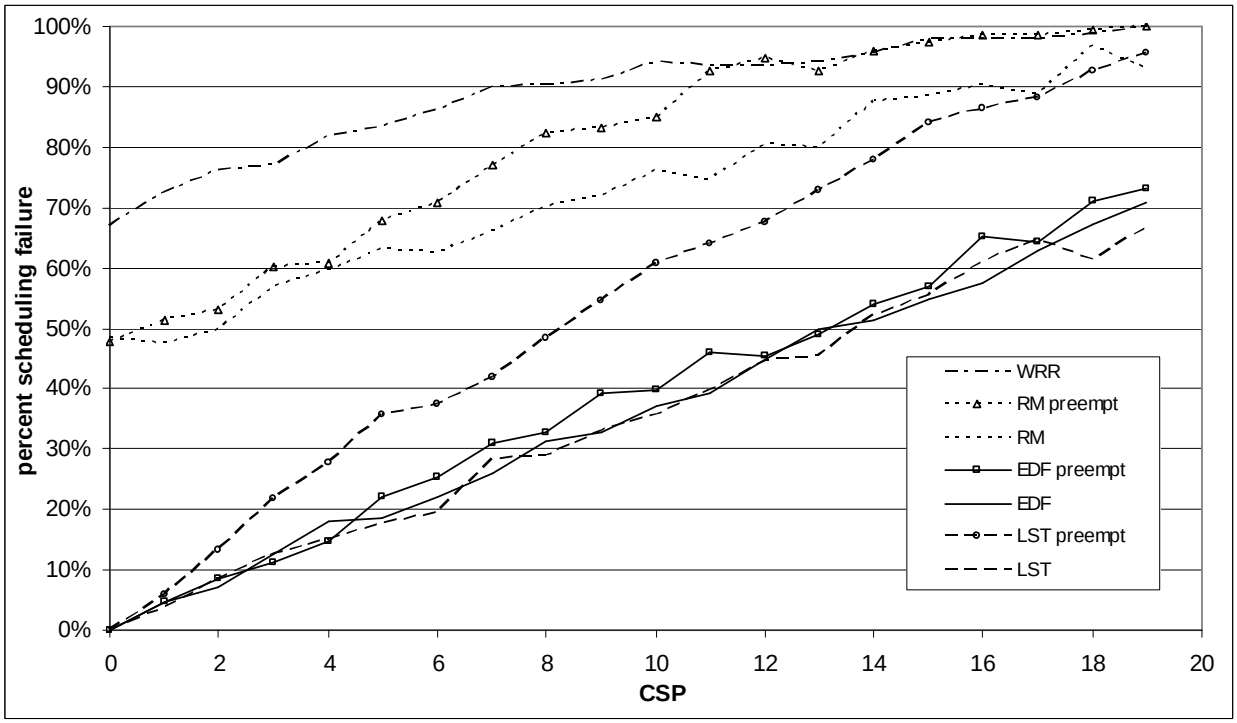


Figure 20: ***Schedulers with context switch penalty, percentage of scheduling failures per context switch penalty***

Increasing either the context switch penalty or the utilization increases the number of scheduling failures. Figure 19 shows the effect of utilization increase on the algorithms when there is a context switch penalty. The distribution of workloads over context switch penalty-parameter is uniform from 0 to 19 (in units analogous to 50 microseconds). A scheduler's performance improves when there is no preemption because no preemption implies fewer context switches, which implies less applications of the context switch penalty. The order of performance in this chart is as follows:

1. EDF scheduler without preemption, LST scheduler without preemption
2. EDF scheduler with preemption, LST scheduler with preemption and RM scheduler
3. The RM scheduler with preemption
4. the WRR scheduler

Figure 20 shows the performance of schedulers per context switch penalty. The order of the schedulers' performance in this chart is:

1. EDF scheduler without preemption, LST scheduler without preemption and EDF scheduler with preemption
2. LST scheduler with preemption
3. RM scheduler
4. RM scheduler with preemption
5. WRR scheduler

6.4.2 Theoretical optimality of EDF and LST scheduling algorithms

The LST scheduler performs worse than the EDF when context switch penalty increases as shown in figure 20 because the LST scheduler switches more. Both the EDF and LST scheduling algorithms are theoretically optimal when preemption is included and there is no context switch penalty. This accounts for their similar performance on the chart varying the context switch penalty (figure 20) when the context switch penalty equals zero.

The following theorems elaborate on the optimality of the EDF and LST schedulers assuming preemption and no context switch penalty. The basic premise of a workload having a feasible schedule is that the utilization is less than or equal to one. If a scheduler can schedule any workload with utilization less than or equal to one, then for realistic expectations, that scheduler will always schedule.

Theorem 6.4.a: If workload W has a feasible schedule, then the EDF scheduler will schedule (proof of Theorem 6.4.a reproduced from "Real-time Systems" [8]).

Suppose parts of W_i and W_k are scheduled in intervals I_1 and I_2 respectively (where W_i and W_k are any two tasks of W). Furthermore, the deadline, d_i of W_i is later than the deadline d_k of J_k, but I_1 is earlier than I_2 as shown in figure 21.

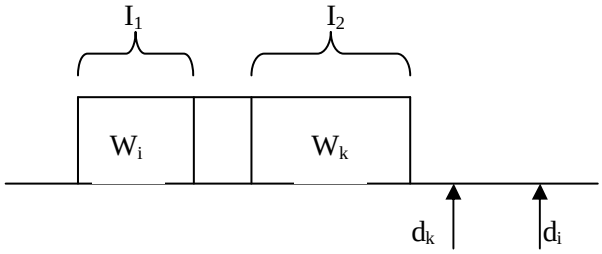


Figure 21: ***Theorem 6.4.a example, initially***

- Case 1: the release time of W_k is later than the end of I_1 and W_k cannot be scheduled in I_1. The two tasks are already scheduled on EDF basis in these intervals.
- Case 2: The release time R_k of W_k is before the end of I_1 (without loss of generality, we assume R_k is no later than the beginning of I_1). Transform this schedule by swapping W_i and W_k in either of the following two ways:
  1. If interval I_1 is shorter than I_2, move the portion of W_k that fits in I_1 to I_1 and move entire portion of W_i scheduled in I_1 to I_2 and place it after W_k (figure 22).

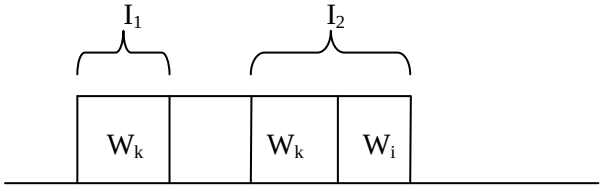


Figure 22: ***Theorem 6.4.a example, first interval shorter***

  2. If interval I_1 is longer than I_2, move entire portion of W_k scheduled in I_2 to I_1; place W_k portion before W_i in I_1 and move the portion of W_i that fits in I_2 to I_2 (figure 23).

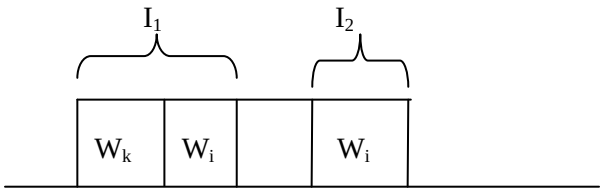


Figure 23: ***Theorem 6.4.a example, first interval longer***

Theorem 6.4.b: If workload W has a feasible schedule using the EDF scheduler, then the LST scheduler will schedule. Proof (slack equals deadline minus remaining execution time):

Suppose parts of W_i and W_k are scheduled in intervals I_1 and I_2 respectively (where W_i and W_k are any two tasks of W). Label the ends of intervals I_1 and I_2 as A and B, respectively (figure 24).

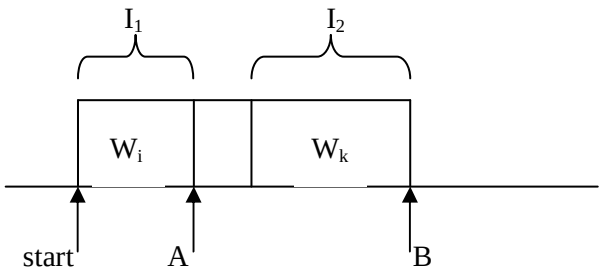


Figure 24: ***Theorem 6.4.b example, initially***

- Case 1: The deadline, d_i of W_i occurs earlier than I_2. In this case, a feasible EDF schedule is the same as an LST schedule.
- Case 2: The deadline, d_i of W_i occurs during I_2. Note that d_k occurs later than or equal to B (we assumed this schedule was feasible). In this case, transform the schedules by swapping portions of W_i in I_1 with portions of W_k in I_2 that precede d_i.
    - If W_i slack = W_k slack at start, then the tasks will run according to round robin until d_i, after which, only W_k will run (note: if d_i = B then W_k may complete first, but the processor will not idle in interval I_2). The length of the portions swapped and the length of time between swapped portions when running round robin will be equal to the LST scheduler's minimum task commitment time.
    - Otherwise, whichever task has less slack at start will run until slacks are equal or until d_i. If slacks become equal, then the tasks will run according to round robin until d_i, after which, only W_k will run.
- Case 3: The deadline, d_i of W_i occurs later than B. If W_i slack = W_k slack at start, then tasks will run according to round robin. Otherwise, whichever task has less slack at start will run until slacks are equal, after which, the tasks will run according to round robin until B.

Theorem 6.4.c: If workload W has a feasible schedule, then the LST scheduler will schedule. This theorem follows by transitivity from theorems 6.4.a and 6.4.b.

7 Published results on the rate monotonic scheduling algorithm

The first sub-section describes two results on the RM scheduling algorithm from separate publications and how they are consistent with my experiments' results. In the second sub-section, I show the distributions over the parameters for the workloads generated in the experiments that have uniform period distributions. The distributions of these experiments show bounds on the HC1 parameter for different distributions. In the third sub-section, a connection between the published results and the HC1 is established. Then I elaborate on how the HC1 and HC0 metrics are actually a generalization of these published results.

7.1 Showing consistency of experiments with published results on the RM scheduling algorithm

The next two sub-sections give a description of two published results on the RM scheduling algorithm and show how this project's experiments are consistent with the papers' data and remarks.

7.1.1 RM scheduling algorithm result, asymptotic performance[3] and distribution of periods

The following four experiments are consistent with a published result on how the distributions of periods have an asymptotic effect on minimum utilization for meeting scheduling deadlines. According to results in "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior" [3], with periods uniformly distributed between 1 and 10, the performance of the RM scheduling algorithm converges to a utilization of 0.814. Figure 25 reproduces a table from [3] with the results for periods uniformly distributed between one and B. The corresponding utilizations are in the column, "Asymptotic performance."

B	Asymptotic Performance
1.0	1.000
1.5	0.811
2.0	0.693
3.0	0.735
5.0	0.773
10.0	0.814
20.0	0.844
80.0	0.867
100.0	0.885

Figure 25: ***Limiting performance of the RM scheduling algorithm, workload periods uniformly distributed*** **[3]**

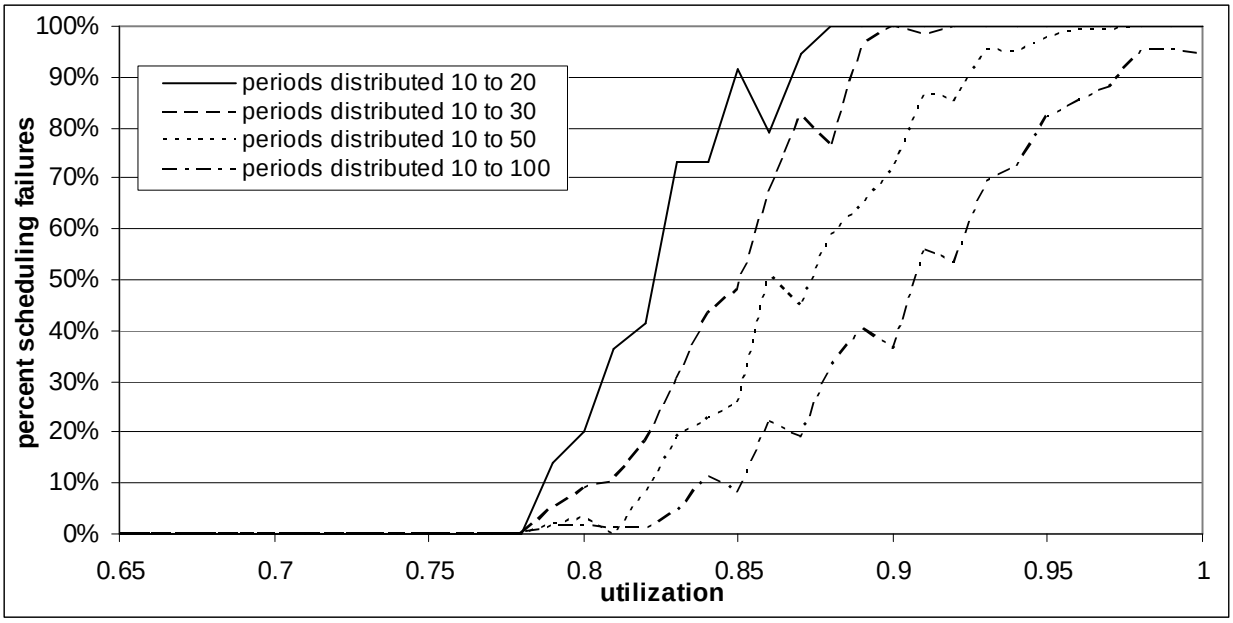


Figure 26: ***RM scheduler with preemption, periods uniformly distributed, percentage of scheduling failures per utilization***

The experiments in Figures 26 and 27 show simulations of the RM scheduler with preemption on workloads that have task periods with the following uniform distributions: 10 to 20, 10 to 30, 10 to 50, and 10 to 100 (analogous to 1 to 2, 1 to 3, 1 to 5 and 1 to 10). Figure 26 shows the how increasing the utilization has a delayed effect of causing scheduling failures depending on the period range in uniform period distributions of tasks. A greater range in the distribution causes a greater delay in scheduling failures. Although these results do not reproduce the same values as the asymptotic performance table (0.693, 0.732, 0.773, 0.814), the fact that the schedulable utilization increases with the same relationship shows consistency with the published results from "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior" [3].

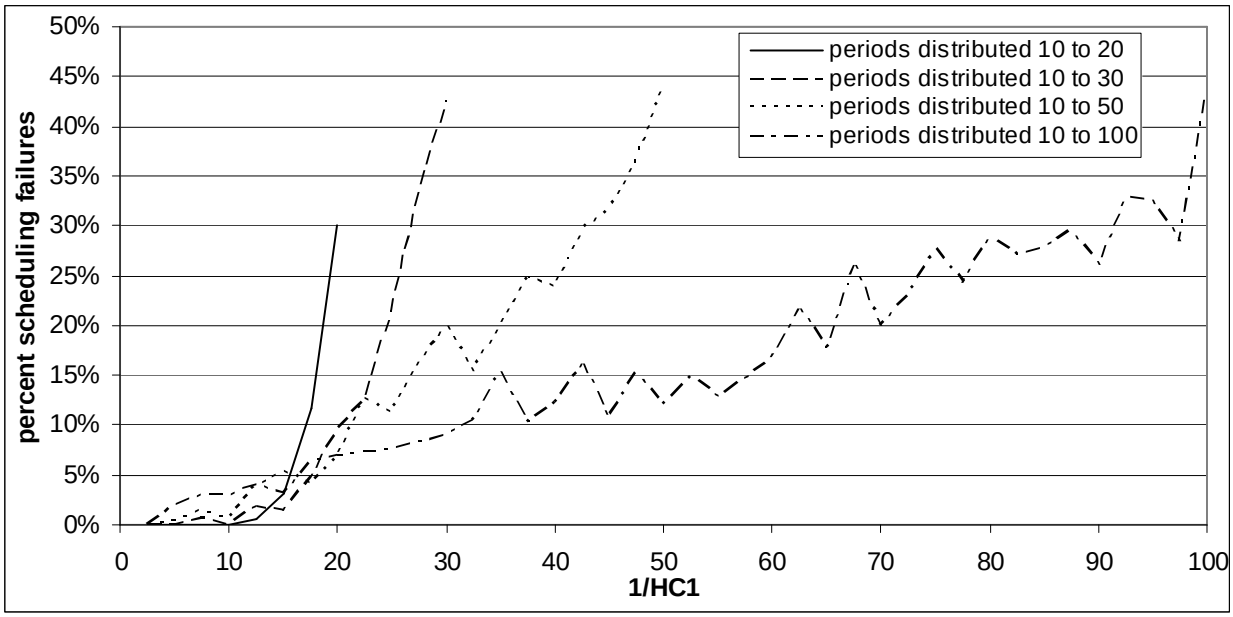


Figure 27: ***RM scheduler with preemption, periods uniformly distributed, percentage of scheduling failures per 1/HC1***

Figure 27 shows the effect of 1/HC1 increase on the RM scheduler with preemption for workloads randomly generated with uniform period distributions. Figure 27 also shows that the percentage of scheduling failures increases with a regular slope until the maximum 1/HC1 value is reached for that range of periods.

7.1.2 RM scheduling algorithm result, utilization bounds and harmonic similarity of tasks

The following result is from the paper, "A Survey of Real-Time Operating Systems – Preliminary Draft" [1]. The authors state that the rate monotonic and other static priority algorithms "can schedule a set of tasks to meet their deadlines if total resource utilization is lower than a schedulable bound." The schedulable bound is "the maximum CPU utilization for which the set of tasks can be guaranteed to meet their deadlines." This bound is "0.693 for RM algorithms in general (with task set size approaching infinity), 0.88 when the periods are uniformly distributed and 1.0 only when the periods are harmonics of the smallest period."

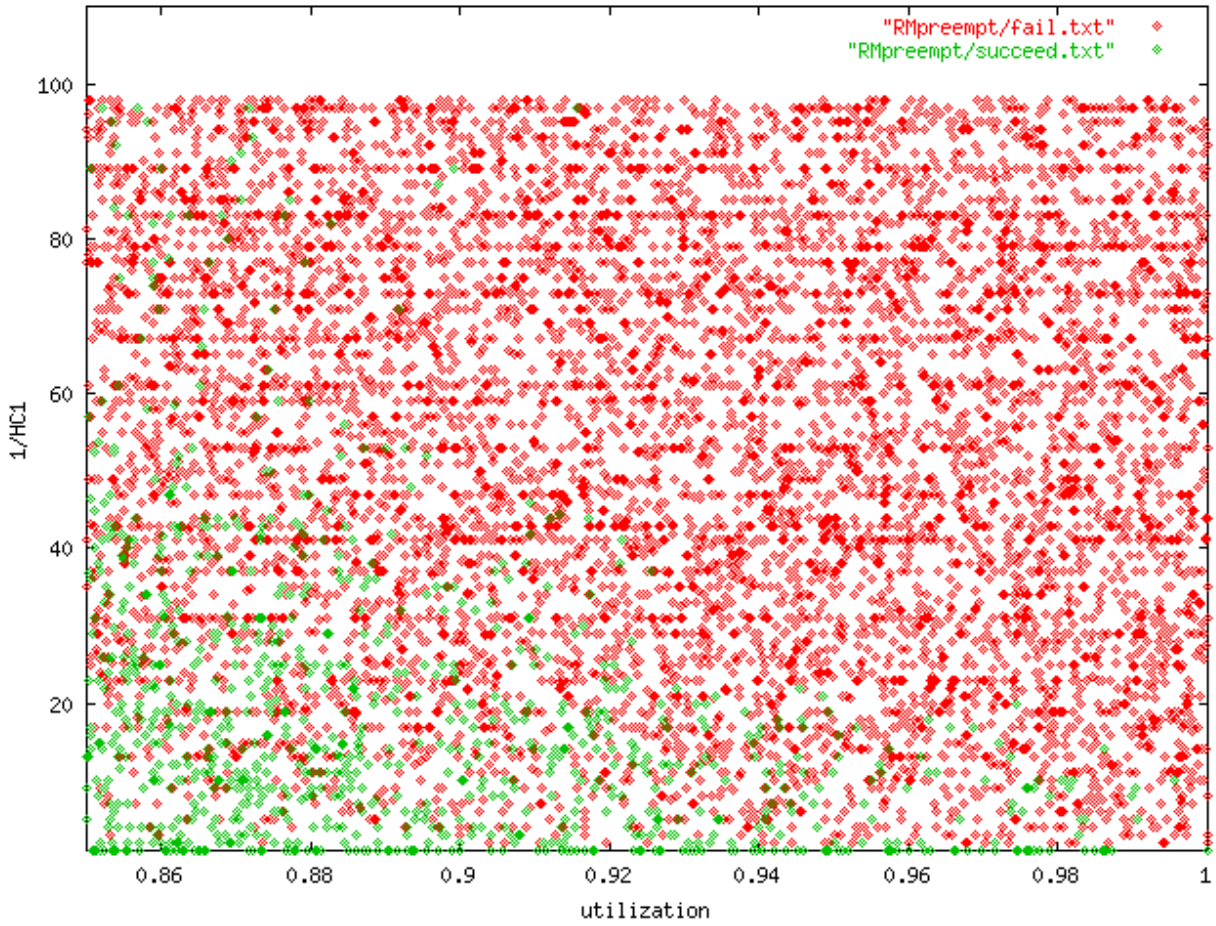


Figure 28: ***RM scheduler with preemption, 1/HC1 vs. utilization (range 0.85 to 1)***

Figure 28 shows the effect that HC1 and utilization have on scheduling for utilization range 0.85 to 1.0. A scheduling success is plotted as a green data point and a failure as a red data point. Whenever task periods are harmonics of the smallest period, the largest period is the LCM of all periods and 1/HC1 equals 1.0. The RM scheduler will always successfully schedule workloads with task periods that are harmonics of the smallest period "A Survey of Real-Time Operating Systems – Preliminary Draft" [1]. Figure 28 supports this claim by the line of success data points in the 1/HC1 = 1.0 row (bottom row of chart). Less harmonically similar task sets exist in the set where 1/HC1 = 1.0. Apparently, all of these task sets were also schedulable. As a workload's periods cease to exhibit harmonic similarity, 1/HC1 will increase. The 0.88 schedulable bound for uniform distribution of periods implies that in the region of the chart (figure 28) between utilizations 0.88 and 1.0, the number of failures will increase as 1/HC1 increases, which it does.

The statement that the RM scheduler schedules all task sets with utilization not greater than 0.693 [3] is supported by data in the chart showing all experiments (figure 8, previous sub-section), but the principle is only illustrated between utilizations 0.775 and 1.0 because no workloads failed to schedule with utilization less than 0.775. The statement about the general case implies that a distribution of workloads with less harmonic similarity than uniform period distributions will allow utilization higher than 0.693 but less than 0.88. This has already been shown to be the case by the existence of the HC1-utilization relationship in the previous section. As was previously the case, for utilization less than 0.8, the failures become too sparse for the relationship to continue.

7.2 Distribution of simulations over parameters, random workloads with uniform period distributions

In figures 29 and 30, the number of workloads per 1/HC1 and utilization for randomly generated workloads having uniform period distributions is shown. Comparing against the distributions from workloads used in the previous experiments, the number of experiments does not have as uniform of a distribution. It is unsurprising though, since the workloads of the previous experiments were generated according to target parameters. The distribution of task periods is uniform from 10 to 20, 30, 50 and 100 as indicated on each of the charts.

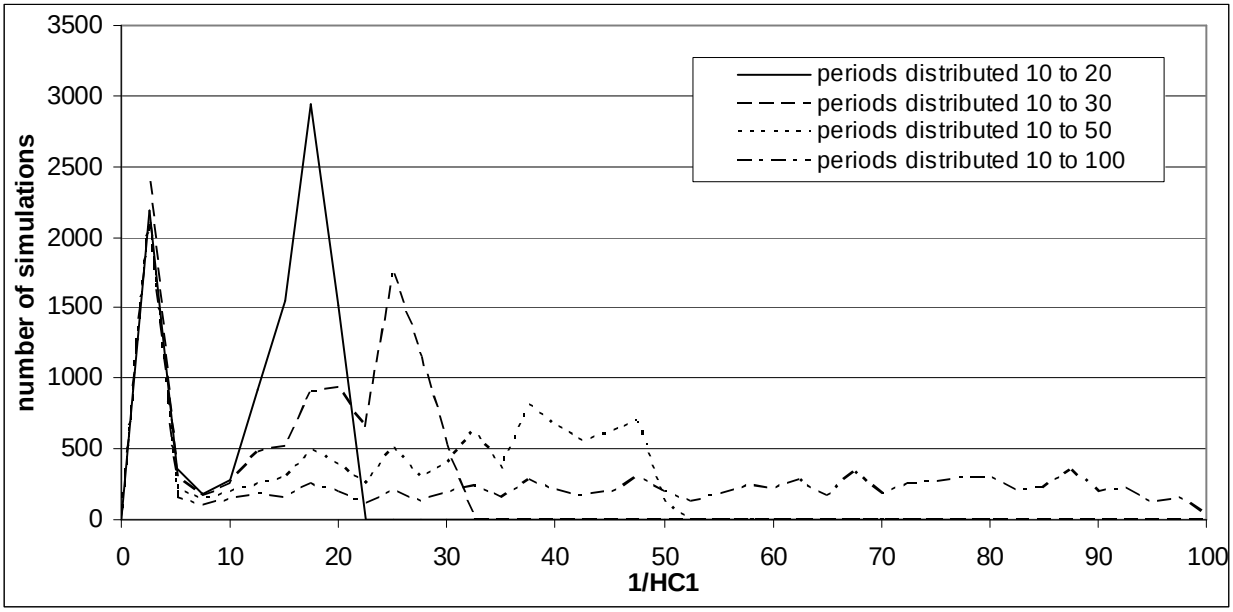


Figure 29: ***RM scheduler with preemption, periods uniformly distributed, simulations per 1/HC1***

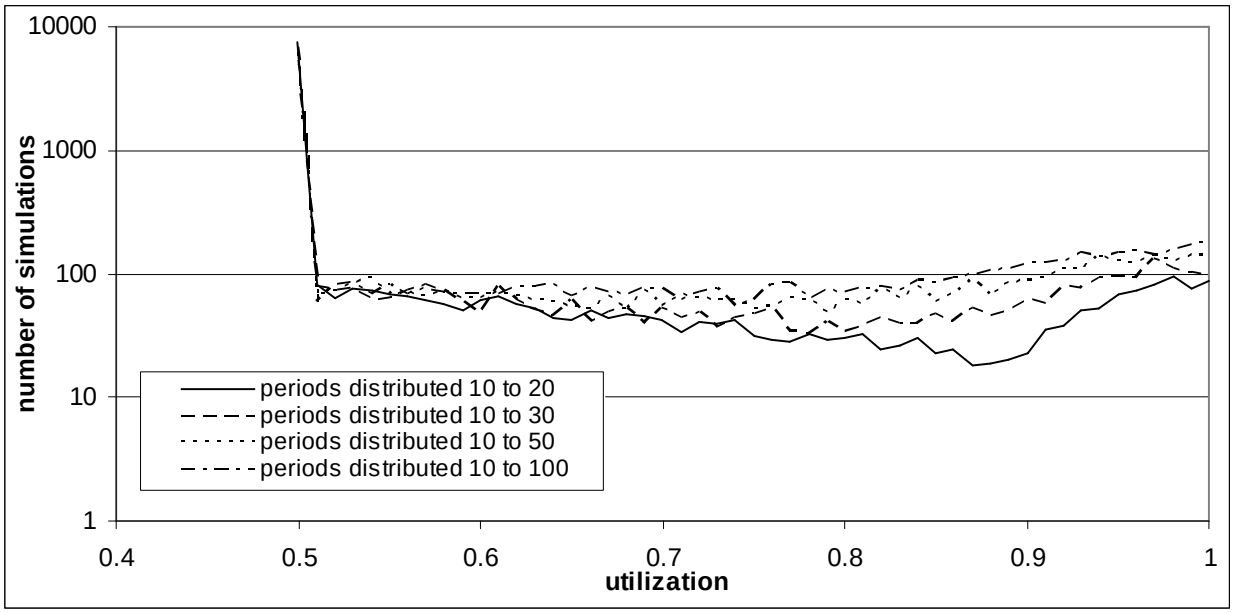


Figure 30: ***RM scheduler with preemption, periods uniformly distributed, simulations per utilization***

Weak uniformity of workload distribution was obtained over utilization for values of utilization above 0.55 (as figure 30 illustrates). We have shown in figure 26 (previous sub-section) that range of task periods affects the schedulability of the RM scheduler on workloads with periods uniformly distributed. Since there is no less than 20 simulations for each value of the utilization in figure 30, confidence level is good for results in figure 26.

7.3 Generalization on published RM scheduling algorithm results using parameters, HC0 and HC1

The results from "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior" [3] indicate a relationship with the range of task periods in workloads with uniform distribution of periods and the performance of the rate monotonic scheduling algorithm. Specifically, the attainable utilization increases as the range of task periods increases (with the exception of range increases when the upper bound is less than twice the lower bound, i.e., period distributions from 1.0 to 1.5). For example, this relationship claims that workloads with periods uniformly distributed between 10 and 20 (or 1 and 2) are less schedulable than workloads with periods uniformly distributed between 10 and 30 (or 1 and 3). Figure 29 shows bounds on the distribution of the task periods impose bounds on the HC1 parameter. Notice that the upper bound on the HC1 parameter increases from 20 to 30 when the upper bound on periods increases from 20 to 30.

In addition, previous results have shown the HC1-utilization relationship is a good predictor for the attainable utilization of a workload. Figure 29 shows that the majority of workloads have an increasing 1/HC1 value, filling in the upper region of the bound, with each successive increase of the upper bound for the periods. The HC1-utilization relationship indicates that distributions of workloads filling in this upper region will increase in average utilization for schedulability. Therefore, the HC1-utilization relationship uses similar characteristics of workloads to draw similar results about schedulable utilization as the relationship with period distribution indicated in "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior" [3].

The three sets as defined in "A Survey of Real-Time Operating Systems – Preliminary Draft" [1] that exemplify the relationship between harmonic similarity of task periods and utilization are: the workload has periods that are 1) "harmonics of smallest period," 2) "uniformly distributed" and 3) "in general" (worst case). Their presence can be verified and their corresponding utilization bounds observed. The range of HC1 and HC0 extends beyond three values however. This project shows that these three sets are merely three data points of a curve that characterizes harmonic similarity of tasks in a more general way. This can be seen in figure 8. In addition, the relationship is shown by average case results in figure 10.

8 Description of visualization system

The visualization system was designed to be an active, modified version of the Gantt chart, a standard in processor scheduling. Figure 31 shows an example of a Gantt chart. First, I describe the Gantt chart. Then, I describe how the visualization system contrasts from it and show screen shots from the visualization system running on example workloads.

The Gantt chart (Figure 31) shows two tasks A and B. The task is displayed as a rectangle on a line. The line represents time for the processor. The rectangle represents the task and the width of the rectangle is the length of time that the task is using the processor. The advantage of a Gantt chart is that it is easy to show how a correct schedule for a workload has only one task using the processor at any given moment.

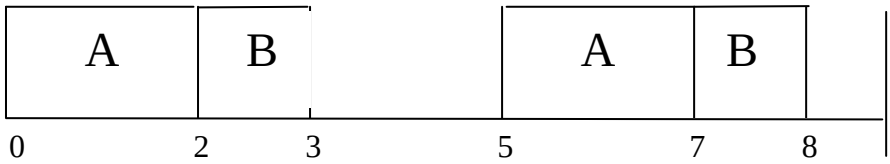


Figure 31: ***Gantt chart***

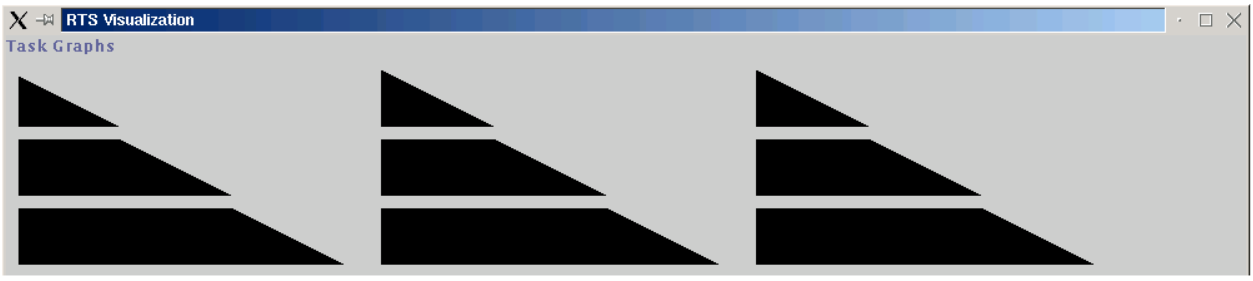


Figure 32: ***Screenshot 1 of visualization system***

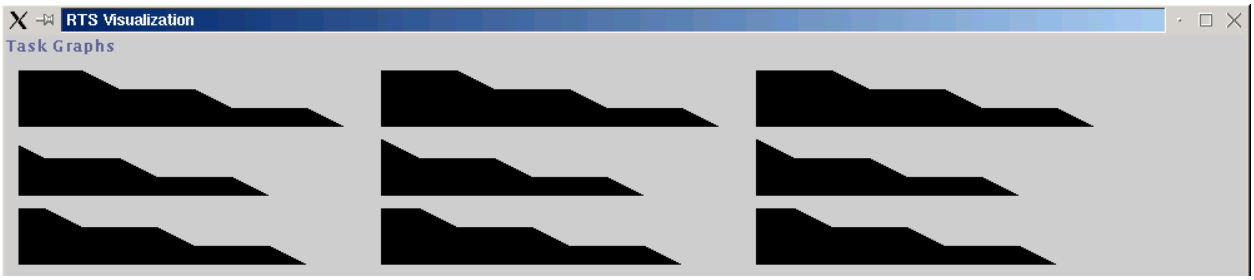


Figure 33: ***Screenshot 2 of visualization system***

The visualization system has a separate line for each task to indicate processor usage. This aids in understanding the workload as a whole. Figures 32 and 33 show screen shots of the visualization system displaying the EDF and RR schedulers, respectively. The amount of remaining work for each task is shown on the visualization system as the height of a shape on the line. As the task executes, the height of the shape decreases at an angle indicating the amount of

time remaining for this task to execute has decreased. The most important feature of the visualization system is to determine the order in which the tasks execute. Another important feature is to show how (and if) the processor moves between tasks before their completion.

9 Conclusion

The results of two papers on the RM scheduling algorithm: "A Survey of Real-Time Operating Systems – Preliminary Draft," [1] and "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior" [3] are consistent with the experiments. The harmonic similarity of task periods is formulated in two ways: HC0 and HC1. The HC0 and HC1 formulas achieve a generalization of published results on RM scheduling algorithm. The experiments show that with a uniform distribution of utilization the average 1/HC1 for failures increases as utilization decreases from 1 to 0.75. For workloads with increasing HC0 and HC1, my experiments show that the percentage of failures increases.

The HC0 and HC1 formulas are applicable results. There are situations in real-time systems and scheduling when exhaustive testing is not feasible. The time required to test every possible combination is beyond the resources of most system-development projects. In these situations, it becomes desirable to test or analyze only using the most adverse conditions that the system would encounter. Scheduling failures under less adverse conditions can be ruled out. The HC0 and HC1 metrics indicate scheduling difficulty that is independent of utilization. Testing and verifying a scheduler for a particular utilization should consist of workload tests for the utilization for a representitive population of HC1 or HC0 values. Writing benchmarks for real-time systems is similar to testing because it also requires a control over scheduling difficulty that HC0 and HC1 provide. If it is beneficial for a system to use a different scheduling algorithm at different points in the system's lifetime, the system could obtain insight as to which algorithm is preferred for the current workload based on an HC0 or HC1 computation. The system could then change scheduling algorithms "on the fly."

The visualization system was very useful in presenting and contrasting the various scheduling algorithms. It was also valuable as a debugging tool both for the implementation of the algorithms and for creating and modifying the simulator. I was fortunate to have written the code for the visualization system early on and used it for debugging throughout the development of the project.

The visualization tool allowed me insight to the scheduling problem and design of experiments. For data analysis, the graphical display of the visualization system was not as useful as spreadsheets and other data collection and interpretation means because of the lack of numerical values. The numbers could be included in the visualization system as well as gridlines and options such as check boxes that would represent data differently. This may help the visualization system in this respect but it would be difficult to get the level of data analysis that is obtainable with modern spreadsheet programs without adding substantially complicated code to the visualization system. If it is clear what the charts for a real-time simulation experiment are required to look like, a visualization system could be designed that displays information for a distribution of simulations. This type of application may be very useful for single-purpose experimentation.

Lastly, the user manual describes all the details necessary for a user to run the simulator on a number of workloads. There also is information to assist a programmer to modify the experiment driver or simulator for use in another project.

10 References

[1] Ghosh, K., Mukherjee, B., Schwan, K. “A Survey of Real-Time Operating Systems – Preliminary Draft,” *Technical Report Georgia Tech. College of Computing*, Report No. GIT-CC-93-18.

[2] Liu, C. L., Layland, J. W. “Scheduling algorithms for multiprogramming in a hard real-time environment,” *Journal of the ACM*, 20(1):46-61, January 1973.

[3] Lehoczy, J., Sha, L., Ding, Y. “The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior”, *Proceedings of 8th IEEE Real-Time Systems Symposium*, pages 166-171, IEEE Computer Society Press, December 1989.

[4] Laplante, Phillip A. “Real-Time Systems Design and Analysis: An Engineer's Handbook”, *Wiley-IEEE Press*, 2nd edition, March 2001.

[5] KURT-Linux: Kansas University Real-Time Linux (website) on 7/11/2003 obtained from: http://www.ittc.ku.edu/kurt/.

[6] NIST Special Publication (Draft-yet to appear), Technology Administration, "Requirements for Real-time Extensions for the Java Platform" *Real-Time for Java Expert Group*, on 7/8/2003 obtained from: http://www.itl.nist.gov/div897/ctg/real-time/intro.html.

[7] Dinkel, W., Niehaus, D., Frisbie, M., Woltersdorf, J. “KURT-Linux User Manual”; on 7/28/2003, obtained from: http://www.ittc.ku.edu/kurt/.

[8] Liu, Jane W. S, “Real-time Systems,” *Prentice-Hall*, 2000.

[9] Silberschatz, A., Galvin, P., “Operating System Concepts,” fifth edition, *Addison Wesley Longman*, 1998

[10] Hennessy, J., Patterson, D., “Computer Architecture, A Quantitative Approach,” third edition, *Morgan Kaufman*, 2003

[11] Knuth, Donald, E., “Art of Computer Programming, volume 2, Semi-Numerical Algorithms," third edition, *Addison Wesley Longman*, 1998

Appendix A, User manual for real-time simulator

Please note there are six ways for running the real-time simulator and/or visualization system outlined in the next two sub-sections ("functional specification" and "commands entered by user").

1. Directories, contents

The software resides in a directory structure reflecting JAVA packages. A diagram of an example directory structure is shown in figure 34 and description of their contents in figure 35.

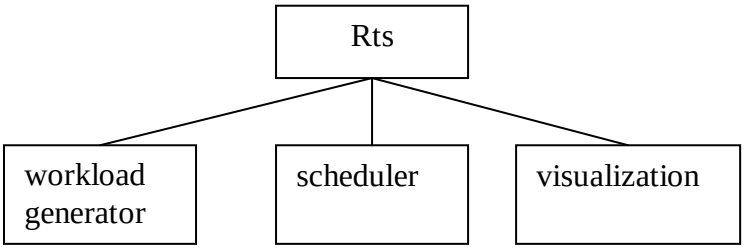


Figure 34: ***Directory structure maintained by the user***

Directory name	Contents
application	data files for example workloads, application software
scheduler	Rts and scheduling algorithms, experimental drivers used in experiments
visualization	visualization software, log files (output of Rts, used by visualization application in playback mode)

Figure 35: ***Description of contents of directories***

2. Commands entered by user

The following are script commands and methods for creating custom commands for running the simulator and visualization system.

2.1. Script for running the simulator and visualization system from a data file:

r <n> <l> <p> <cs> <id>

<n> id number for workload file (n may be a number or a character) file pn.txt

<l> (optional*) length of simulation
default is 100

<p> (optional*) preemption indicator, if this number is any number other than zero, the simulation will run with preemption. The default is no preemption

<cs> (optional*) context switch penalty, time added to the elapsed simulation time when no tasks are run and assessed when two tasks are swapped between cycles (in 1/20 seconds)
default is 0 (no context switch penalty)

<id> (optional*) id for the scheduling algorithm
default is 0 (EDF)

- Each of the tasks' remaining work is displayed on visualization system.
- Example workload files exist in application directory. The description of these workload files is in the "Interface 1" sub-section of the "Interfaces" sub-section of the *architectural specification* (Appendix B).

2.2. Script for running just the simulator from a data file:

rts <n> <l> <p> <cs> <id>

- Arguments are defined as above
- Simulation is displayed using text on standard output

2.3. Script for running the simulator and visualization system from a data file and save the visualization information in a log file:

rLog <n> <l> <p> <cs> <id>

- Arguments are defined as above
- All details are the same as the first command except data is also saved in a file: logFile.<simulation-identifier>.txt where the *simulation-identifier* is defined as follows: If the arguments of the simulation are a, 30, 1, 0, and 0 then the *simulation-identifier is*: a.30.1.0.0 and the name of the log file is, logFile.a.30.1.0.0.txt. The log file can be read by human user or used as input to the visualization system in the following command:

2.4. Script for running just the visualization system from a data file:

v < simulation-identifier>

- Command runs the visualization system in playback mode from the corresponding log file (*simulation-identifier* as defined above).

2.5. Script for running the experiment driver:

- optional arguments are defined by experiment

rexp

2.6. Write an experiment driver (or modify the existing driver). The Rts can be run by invoking the runSimulation method.

*note: all optional arguments must be included cumulatively (i.e.: argument three cannot be included without one and two, if an argument is not included, the simulation runs with the default value)

Appendix B, Architectural specification for real-time simulator

The following is an architectural specification consisting of a system overview, software modules, object inheritance, object dependencies and application interfaces.

1. Overview of system

Figure 36 shows an overview of the system's main components.

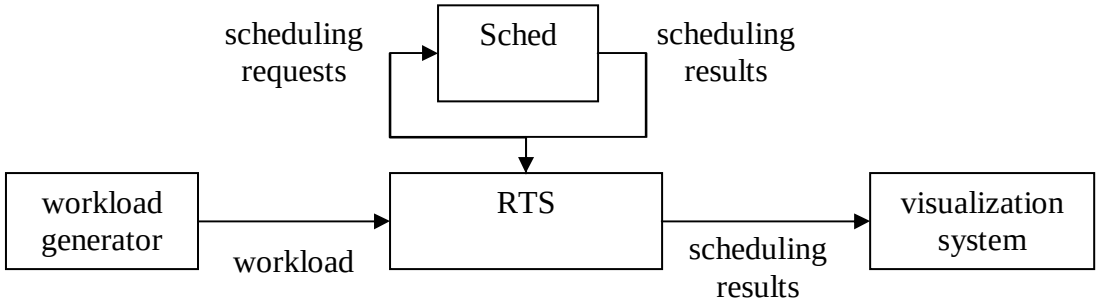


Figure 36: ***System overview***

2. Software modules

The software modules form a library whose components are modifiable for use in a similar software system either in their entirety or individually. All software is written in JAVA. UNIX scripts are written for the C-shell. The software modules in figure 37 are described below.

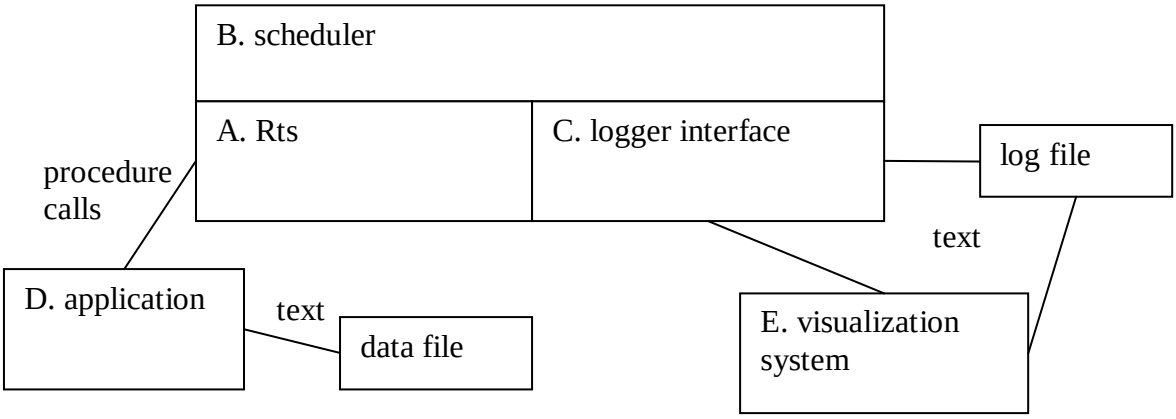


Figure 37: ***Software modules for system***

A. **Rts**: Module contains the wait queue and method to run the simulation

- *runSimulation (int maxSimulationTime, Boolean preemptionEnabled, int contextSwitchPenalty, int schedAlgorithmId, Boolean outputVisualizationData) maxSimulationTime* is the length of simulation. If *preemptionEnabled* = true, then this simulation will enable preemption and vice versa. The *contextSwitchPenalty* is time added to the elapsed simulation time when no tasks are run and assessed when two tasks are swapped between cycles.

Tasks are swapped according to selected *schedAlgorithmId*: 0 = EDF, 1 = RM, 2 = LST, 4 = RR, 5 = WRR. The WRR supports tasks of two priorities, hard real-time and soft real-time. The hard real-time quantum is 3 and the soft real-time quantum is 1. If the *outputVisualizationData* option is set to true, the data of the simulation described as "Interface 2" in the "Interfaces" sub-section of the *architectural specification* (Appendix B) is output to standard output.

B. **Logger interface**: Module consists of a method, outputData(), which outputs the data from the real-time simulation. This data is readable by the visualization system to create the graphical display.

C. **Application**: Module defines the workload for the simulation. Methods in this module are defined as follows:
   - *getInput() called by Rts to create workload from standard input*
   - *genWorkload(utilization) called by RtsDriver, generates tasks according to target utilization*
   - *genWorkload(utilization, HcValue, HcType) called by RtsDriver, generates tasks according to target utilization and target HC0 or HC1 value. If HcType = 0 then HC0 is used. If HcType = 1, HC1 is used.*
   - *getHC(HcType) finds the actual HC0 or HC1 value for a workload (*type defined above)*
   - *getUtilization() finds the actual utilization for a workload*

D. **Scheduler**: Module contains a ready queue, a method for scheduling tasks for execution, and methods allowing the Rts to control when and for how long the scheduler will run. All methods for the scheduler are called from Rts.
   - *scheduleTask (Task t)* A task may be periodic or aperiodic. Periodic tasks repeat requests at a regular interval equal to the period. For simplicity, the period will be equal to deadline – request time. Therefore, requests will repeat in the following fashion: *requestTime*, *requestTime* + *period*, *requestTime* + (2 * *period*), *requestTime* + (3 * *period*)… etc. Aperiodic tasks do not repeat. For definition of *Period* and *RequestTime*, see "Description of Task Fields" below.
   - *GetCurrentRunningTask ()* This method returns the current task to run according to scheduler's priority
   - *findCycleTime ()* Finds the cycle time that will bring the scheduler to the next appropriate time to make a scheduling decision.
   - *updateScheduler (int time)* Runs the scheduler, updates the tasks and the scheduler "time" time units.
   - *updateSchedulerContextSwitch (int time)* updates the tasks and the scheduler "time" time units to account for the context switch penalty that is required for this Rts

E. **Visualization System**: Module displays data directly output by the logger interface of the Rts. The visualization system can also be run as a stand-alone application in which case it plays back the data of a saved log file. The different usages of the visualization application are defined in "commands run by user" sub-section of this user manual.

F. **Task**: Module is a data type used by all modules (fields defined in figure 38).

Field	Description
taskId	Unique integer identifier for task
priority	defines the type of real-time task, 0 = hard, 1 = soft.
execTime	execution time; Worst case time that this task must be allowed to run in order to complete
period	the time between release times for periodic tasks
nextDeadline	time until this task's next deadline
runningTime	time that this task has been allowed to execute
isPeriodic	true = periodic, false = non-periodic
releaseTime	release time of task in seconds relative to zero (beginning of simulation). Also, the time that the simulation must run before this task will become available for execution

Figure 38: ***Description of fields in Task software module***

The experimental driver (RtsDriver) for experiments run in this project is not a part of the design so the depth of coverage is limited. The command to run the driver is described and information about the driver's interface with this software can be found in the description of Rts and Application modules.

3. Object inheritance

All software modules are implemented in JAVA classes, which exhibit the inheritance graph shown in figure 39. Vector is java.util.Vector.

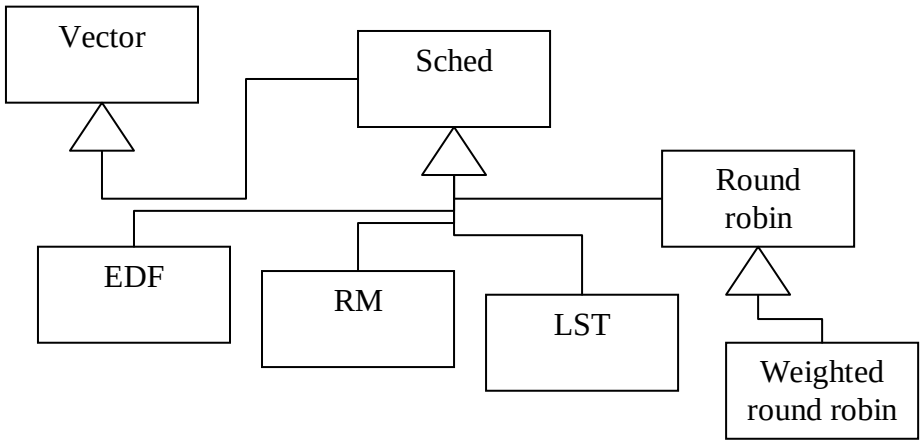
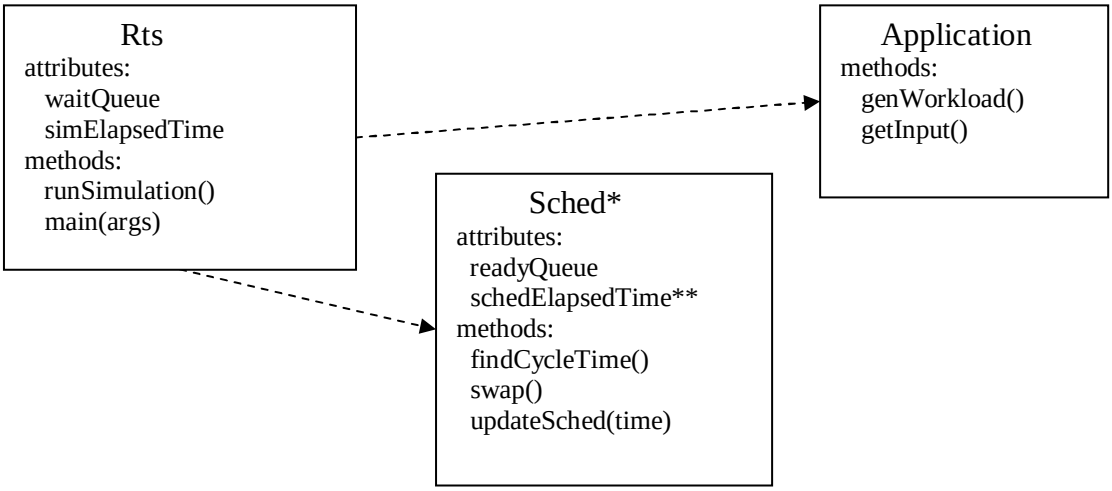


Figure 39: ***Inheritance graph for classes***

The following classes (not shown in Figure 39) of the project are derived from "Object" base class: Vis, Rts, RtsDriver, Application, and Task.

4. Object dependencies

Figure 40 shows the dependency of classes, class fields and class methods. The Visualization system is a separate application from the Rts connected by a standard input output connection. When running experiments, class RtsDriver calls methods from Rts and Application.



All objects use the Task object as a data structure
**The scheduler object is instantiated by Rts and can be any of the subclasses corresponding to algorithms RM, EDF, etc.*
***The scheduler's elapsed time is equal to or less than the Rts because the Rts runs the scheduler on the tasks selected for it.*

Figure 40: ***Object dependencies for classes***

5. Interfaces

The interfaces of the system, displayed in figure 41, are described below. Both interfaces 1 and 2 consist of ASCII text and are implemented using either standard I/O or a file.

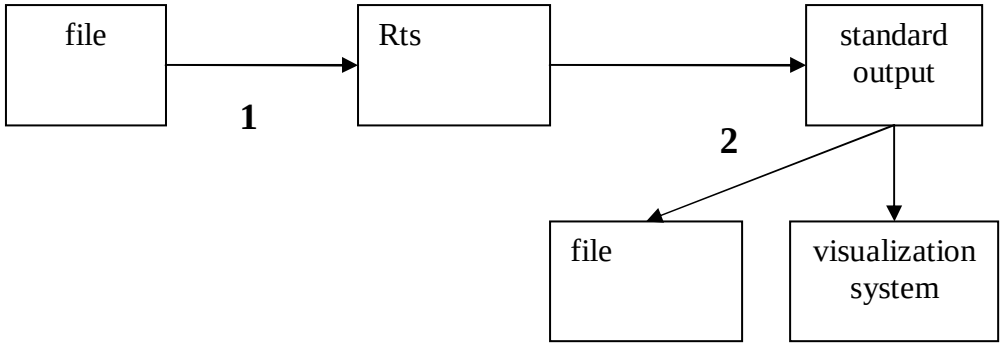


Figure 41: ***Application interfaces***

5.1. Interface 1

The text in this interface describes the tasks for the simulation. For actual text-files (included in this submission) representing this interface, see Appendix C of project final paper.

5.1.1. Interface 1 file format:

The file contains integer task parameters separated by commas and contained in parenthesis (parameters defined in figure 42, format: (id, rt, p, xt, period)).

mnemonic	Max digits	Description
Id	2	Unique integer identifier for task
Rt	3	*releaseTime* (defined in figure 38)
P	1	priority (0, 1, 2, 3) (defined in figure 43)
Xt	2	*execTime* (defined in figure 38)
Period	2	*period* (defined in figure 38)

Figure 42: ***Task parameters in data files***

5.1.2. Interface1 explanation of priorities:

The priority of tasks in the data files is different from priorities in the Application and Rts modules. The data file priority defines both the actual priority and the periodicity of a task. Figure 43 contains a table that defines the values.

	Hard real-time	Soft real-time
Periodic	0	1
Aperiodic	2	3

Figure 43: ***Priorities of tasks in data files***

5.1.3. Description of included workload files:

Figure 44 is a description of the included workload files in table format.

filename	***description***	***Release times, description***
p1.txt	Periodic	0
p2.txt	Periodic	0, utilization = 1.0
p3.txt	Periodic	0
p4.txt	Periodic	0
p5.txt	Periodic	0
p6.txt	Periodic	0
p7.txt	Periodic	0
p8.txt	Periodic	0
p9.txt	Aperiodic	0 and non-zero
pa.txt	Both	0
pb.txt	Both	Aperiodics are non-zero
pc.txt	Both	0 and non-zero Both are non-zero
pd.txt	Both	Aperiodics are non-zero, overwriting of ids
pe.txt	Periodic	long periods (testing RR)
pf.txt	Periodic	SRT and HRT (testing WRR)
pg.txt	Periodic	High utilization, long periods

ph.txt	Periodic	Long periods, many tasks
pi.txt	Periodic	Prime periods, testing HC0/HC1
pj.txt	Periodic	Prime periods, testing HC0/HC1
pk.txt	Periodic	Prime periods, testing HC0/HC1

Figure 44: ***Table of information on included workload files***

5.2. Interface 2

This interface is the output of the simulation. There is a line of text for each time increment of the simulation. The following describes how the scripts included in the project work by manipulating the program input/output.

To run the visualization system, the output of Rts is piped (using a UNIX pipe) to the input of the visualization system. The user reads the numerical data displayed on standard output during simulation without the visualization system. If the rLog command is run, the output of Rts is piped using a T-joint to the visualization system and to a log file. The log file can be read later by the human user or used as the input to the visualization system in playback mode. For an actual text-file representing this interface, see Appendix D of the project final paper.

Some of the data in the log files is not used by the visualization system. This data is analogous to comments. Data contained in parentheses is not read by the visualization system. There is information about the current running task, release of any new tasks and the status and elapsed time of the simulation. The data that is read by the visualization system is contained in brackets and follows the following format:

[r0, r1, r2, r3, r4, r5, r6, r7, r8, r9]

'*rn*' is the remaining seconds of work for task id n.

Explanation of fields (line from example file Appendix D):

```
(20)(t3)  [0, 3, 0, 1, 1, 3, 2, 1, 0, 0] (cxt~id:2 pr:0 nd:9 xt:2 p:9 rt:2 false)
```

`(20)` Amount of time executed this cycle (in 1/20 fractional seconds)

`(t3)` Amount of time elapsed by simulation (in seconds)

`[0, 3, 0, 1, 1, 3, 2, 1, 0, 0]` Remaining work for tasks, read by visualization system

`(cxt~id:2 pr:0 nd:9 xt:2 p:9 rt:2 false)` Details about the currently executing task

Appendix C, Data files

The following input files representing example workloads are included in program archive (the workloads are described in the interfaces sub-section of the *architectural specification*, Appendix B).

```
p1.txt
::::::::::::::
(0, 0, 0, 2, 9)(1, 0, 0, 3, 9)(2, 0, 0, 2, 9)
::::::::::::::
p2.txt
::::::::::::::
(0, 0, 0, 1, 3)(1, 0, 0, 2, 6)(2, 0, 0, 2, 6)
::::::::::::::
p3.txt
::::::::::::::
(0, 0, 0, 1, 4)(1, 0, 0, 2, 6)(2, 0, 0, 2, 6)
::::::::::::::
p7.txt
::::::::::::::
(0, 0, 0, 1, 9)(1, 0, 0, 3, 18)(2, 0, 0, 2, 9)(3, 0, 0, 1, 18)
(4, 0, 0, 1, 18)(5, 0, 0, 3, 27)(6, 0, 0, 2, 22)(7, 0, 0, 1, 14)
::::::::::::::
p8.txt
::::::::::::::
(0, 0, 0, 9, 20)(1, 0, 0, 2, 15)(2, 0, 0, 1, 5)
::::::::::::::
p9.txt
::::::::::::::
(0, 0, 2, 2, 5)(1, 0, 3, 1, 8)(2, 0, 2, 2, 9)
(0, 10, 2, 2, 5)(1, 20, 3, 1, 8)(2, 30, 2, 2, 9)
::::::::::::::
pc.txt
::::::::::::::
(0, 10, 0, 1, 9)(1, 50, 1, 3, 18)
(4, 30, 0, 1, 18)(5, 15, 1, 3, 27)
(2, 40, 2, 2, 9)(3, 60, 3, 1, 18)
(6, 20, 2, 2, 22)(7, 65, 3, 1, 14)
::::::::::::::
pd.txt
::::::::::::::
(0, 10, 0, 1, 9)(1, 50, 1, 3, 18)
(4, 30, 0, 1, 18)(5, 15, 1, 3, 27)
(2, 40, 2, 2, 9)(3, 60, 3, 1, 18)
(6, 20, 2, 2, 22)(7, 65, 3, 1, 14)
(2, 140, 2, 9, 9)(3, 160, 3, 1, 18)
(6, 120, 2, 7, 22)(7, 165, 3, 1, 14)
::::::::::::::
pe.txt
::::::::::::::
(0, 0, 0, 9, 27)(1, 0, 0, 9, 27)(2, 0, 0, 9, 27)
::::::::::::::
pf.txt
::::::::::::::
(0, 0, 0, 9, 27)(1, 0, 0, 9, 27)(2, 0, 0, 6, 27)(3, 0, 1, 6, 27)
::::::::::::::
pg.txt
::::::::::::::
(0, 0, 0, 9, 27)(1, 0, 0, 9, 27)(2, 0, 0, 6, 27)
::::::::::::::
ph.txt
::::::::::::::
(0, 0, 0, 2, 25)(1, 0, 0, 3, 25)(2, 0, 0, 1, 5)
(3, 0, 0, 5, 25)(4, 0, 0, 3, 25)(5, 0, 0, 1, 5)
```

Appendix D, Log file

The following is data collected from an experiment on an example workload (the fields of the output are described in the interfaces sub-section of the *architectural specification*, Appendix B). The length of time chosen by command line option for this simulation was 30.

```
(created task: ~id:0 pr:0 nd:9 xt:1 p:9 rt:0 true)
(created task: ~id:1 pr:1 nd:18 xt:3 p:18 rt:0 true)
(created task: ~id:2 pr:0 nd:9 xt:2 p:9 rt:0 false)
(created task: ~id:3 pr:1 nd:18 xt:1 p:18 rt:0 false)
(created task: ~id:4 pr:0 nd:18 xt:1 p:18 rt:0 true)
(created task: ~id:5 pr:1 nd:27 xt:3 p:27 rt:0 true)
(created task: ~id:6 pr:0 nd:22 xt:2 p:22 rt:0 false)
(created task: ~id:7 pr:1 nd:14 xt:1 p:14 rt:0 false)
(running Simulation for : 600time units)
(Preemption Enabled)
(contextSwitchPenalty: 0)
(schedAlgorithmId: 0)
(  )(t0)  [1, 3, 2, 1, 1, 3, 2, 1, 0, 0] (cxt null)
(20)(t1)  [0, 3, 2, 1, 1, 3, 2, 1, 0, 0] (cxt~id:0 pr:0 nd:9 xt:1 p:9 rt:1 true)
(20)(t2)  [0, 3, 1, 1, 1, 3, 2, 1, 0, 0] (cxt~id:2 pr:0 nd:9 xt:2 p:9 rt:1 false)
(20)(t3)  [0, 3, 0, 1, 1, 3, 2, 1, 0, 0] (cxt~id:2 pr:0 nd:9 xt:2 p:9 rt:2 false)
(20)(t4)  [0, 3, 0, 1, 0, 3, 2, 1, 0, 0] (cxt~id:4 pr:0 nd:18 xt:1 p:18 rt:1 true)
(20)(t5)  [0, 3, 0, 1, 0, 3, 1, 1, 0, 0] (cxt~id:6 pr:0 nd:22 xt:2 p:22 rt:1 false)
(20)(t6)  [0, 3, 0, 1, 0, 3, 0, 1, 0, 0] (cxt~id:6 pr:0 nd:22 xt:2 p:22 rt:2 false)
(20)(t7)  [0, 3, 0, 1, 0, 3, 0, 0, 0, 0] (cxt~id:7 pr:1 nd:14 xt:1 p:14 rt:1 false)
(20)(t8)  [0, 2, 0, 1, 0, 3, 0, 0, 0, 0] (cxt~id:1 pr:1 nd:18 xt:3 p:18 rt:1 true)
(20)(t9)  [1, 1, 0, 1, 0, 3, 0, 0, 0, 0] (cxt~id:1 pr:1 nd:18 xt:3 p:18 rt:2 true)
(20)(t10) [0, 1, 0, 1, 0, 3, 0, 0, 0, 0] (cxt~id:0 pr:0 nd:18 xt:1 p:9 rt:1 true)
(20)(t11) [0, 1, 0, 0, 0, 3, 0, 0, 0, 0] (cxt~id:3 pr:1 nd:18 xt:1 p:18 rt:1 false)
(20)(t12) [0, 0, 0, 0, 0, 3, 0, 0, 0, 0] (cxt~id:1 pr:1 nd:18 xt:3 p:18 rt:3 true)
(20)(t13) [0, 0, 0, 0, 0, 2, 0, 0, 0, 0] (cxt~id:5 pr:1 nd:27 xt:3 p:27 rt:1 true)
(20)(t14) [0, 0, 0, 0, 0, 1, 0, 0, 0, 0] (cxt~id:5 pr:1 nd:27 xt:3 p:27 rt:2 true)
(20)(t15) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (cxt~id:5 pr:1 nd:27 xt:3 p:27 rt:3 true)
(20)(t16) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (cxt null)
(20)(t17) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (cxt null)
(20)(t18) [1, 3, 0, 0, 1, 0, 0, 0, 0, 0] (cxt null)
(20)(t19) [0, 3, 0, 0, 1, 0, 0, 0, 0, 0] (cxt~id:0 pr:0 nd:27 xt:1 p:9 rt:1 true)
(20)(t20) [0, 3, 0, 0, 0, 0, 0, 0, 0, 0] (cxt~id:4 pr:0 nd:36 xt:1 p:18 rt:1 true)
(20)(t21) [0, 2, 0, 0, 0, 0, 0, 0, 0, 0] (cxt~id:1 pr:1 nd:36 xt:3 p:18 rt:1 true)
(20)(t22) [0, 1, 0, 0, 0, 0, 0, 0, 0, 0] (cxt~id:1 pr:1 nd:36 xt:3 p:18 rt:2 true)
(20)(t23) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (cxt~id:1 pr:1 nd:36 xt:3 p:18 rt:3 true)
(20)(t24) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (cxt null)
(20)(t25) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (cxt null)
(20)(t26) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (cxt null)
(20)(t27) [1, 0, 0, 0, 0, 3, 0, 0, 0, 0] (cxt null)
(20)(t28) [0, 0, 0, 0, 0, 3, 0, 0, 0, 0] (cxt~id:0 pr:0 nd:36 xt:1 p:9 rt:1 true)
(20)(t29) [0, 0, 0, 0, 0, 2, 0, 0, 0, 0] (cxt~id:5 pr:1 nd:54 xt:3 p:27 rt:1 true)
(20)(t30) [0, 0, 0, 0, 0, 1, 0, 0, 0, 0] (cxt~id:5 pr:1 nd:54 xt:3 p:27 rt:2 true)
(rts execution complete)
```

Appendix E, Description of deliverables

1. Printed
   A. Documentation
      1. Final Report (sections 1-9, Appendices A-E)

2. Included on CD-ROM
   A. Program archive
      1. Source code
      2. Scripts (see “commands entered by user” in user manual for a functional description of scripts included in the archive)
      3. Javadoc from code
   B. Documentation in PDF format
      1. Final Report (sections 1-9, Appendices A-E)
      2. Presentation slides from defense (separate file)

- charts created using Microsoft Excel and Gnuplot (open-source version)
- documentation typeset in Microsoft Word