A RAND NOTE

Non-Preemptive Time Warp Scheduling Algorithms

Christopher Burdorf, Jed Marti

June 1990

RAND

This research was sponsored by the United States Army under Contract No. MDA903-86-C-0059.

Published by The RAND Corporation
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138

A RAND NOTE

N-3099-A

Non-Preemptive Time Warp Scheduling Algorithms

Christopher Burdorf, Jed Marti

June 1990

Prepared for the
United States Army

RAND

PREFACE

The Concurrent Processing for Advanced Simulation project developed programming systems for large-scale simulations. This paper outlines experiments that explore the effect of multiprocessing scheduling algorithms on the overall performance of Time Warp based simulations. This paper will be of interest to programmers and designers of multiprocessor systems.

The work has been conducted in the Applied Technology Program of the Arroyo Center. Questions involving technical issues should be addressed to Dr. Jed Marti, Project Leader.

The Arroyo Center

The Arroyo Center is the U.S. Army's Federally Funded Research and Development Center for studies and analysis operated by The RAND Corporation. The Arroyo Center provides the Army with objective, independent analytic research on major policy and management concerns, emphasizing mid- to long-term problems. Its research is carried out in five programs: Policy and Strategy Studies; Force Development and Employment; Readiness and Sustainability; Manpower, Training, and Performance; and Applied Technology.

Army Regulation 5-21 contains basic policy for the conduct of the Arroyo Center. The Army provides continuing guidance and oversight through the Arroyo Center Policy Committee, which is co-chaired by the Vice Chief of Staff and by the Assistant Secretary for Research, Development, and Acquisition. Arroyo Center work is performed under contract MDA903-86-C-0059.

The Arroyo Center is housed in RAND's Army Research Division. The RAND Corporation is a private, nonprofit institution that conducts analytic research on a wide range of public policy matters affecting the nation's security and welfare.

Stephen M. Drezner is Vice President for the Army Research Division and Director of the Arroyo Center. Those interested in further information concerning the Arroyo Center should contact his office directly:

Stephen M. Drezner
The RAND Corporation
1700 Main Street
P.O. Box 2138
Santa Monica, California 90406-2138
Telephone: (213) 393-0411

Acknowledgments

We would like to thank John Fitch and Paul Steinberg for reviewing this paper and providing helpful comments and suggestions.

SUMMARY

The Time Warp multiprocessing scheme promises speedup for object-oriented discrete-event simulations. The Concurrent Processing for Advanced Simulation project has constructed a Lisp-based Time Warp system for implementing simulations with many large, complex objects. Since many objects share a single processor, the CPU time allocated to each object must be scheduled. Since object events are not preempted, we are scheduling which objects have events processed rather CPU time per object. We developed approaches to scheduling, ranging from a simple Round-Robin mechanism to complex ones involving queue length.

We developed ten different scheduling algorithms which we named: Worst Case, Conventional Round-Robin, Lowest Local Virtual Time (LVT) First, Priority LVT, Largest Queue Priority, Bradford/Fitch, Anti-Penalty, Queue Anti-Penalty, Queue Cycle, and Positive Infinity.

Results show that LVT, anti-messages, rollbacks, returned messages, and anti-reminders are good parameters for scheduling of system resources. Input queue size is also an important factor, but when taken with or without LVT, it does not produce results as good as using LVT alone. The round-robin scheduler was one of the worst performers. The poor performance of the simple round-robin scheduler indicates the advantages of using state information to determine the scheduling order in the Time Warp system.

Benchmarks of the schedulers showed that the Anti-Penalty scheduler performed better than the others. The Anti-Penalty algorithm is based on a composite measure of simulation advance rate, flow control, and the appearance of specific message types. The benchmark simulation executed on a five processor Time Warp system.

CONTENTS

LIST OF FIGURES

1 Introduction

The Time Warp multiprocessing scheme promises speedup for object-oriented discrete-event simulations. RAND's Concurrent Processing for Advanced Simulation project has constructed a Lisp-based Time Warp system for implementing simulations with many large complex objects. Time Warp is an optimistic strategy for distributed simulation that distributes objects and workload across nodes in a network of workstations. This paper describes a study of the scheduling algorithm used to share the same processing unit among a number of objects[1].

To minimize storage fragmentation, our multiprocessing system operates as a single job per processor. The system allocates dynamic storage from a single heap shared by a processor's resident objects. The task scheduler allows each object to execute for a a number of simulation event cycles before switching contexts to another. Our experiments vary the number of cycles, the frequency an object is selected for execution, and the order of object selection. Our goal was to identify parameters that play a major role in Time Warp scheduling and to test various mechanisms that improved its performance.

After describing the general class of Time Warp systems, we review the different classes of distributed schedulers and argue for the existence of a new class. We then describe the different variants of schedulers we tested, the results, and then close with a discussion of the costs and benefits of using different features of the state to improve performance.

2 The Time Warp Mechanism

Time Warp is an object-oriented message-passing scheme for transparent multiprocessing [4, 6]. It is particularly well suited for object-oriented discrete-event simulations with large numbers of objects having complex behaviors. Objects require a large amount of code to define their behavior. Time Warp differs from single processor discrete-event simulations in three key ways. First, while conventional discrete-event simulations have a single simulation clock serving all objects, each Time Warp object has its own clock. Second, in a conventional discrete-event simulation, all objects execute with the same simulation time, whereas in a Time Warp simulation, each object proceeds at its own rate. Third, conventional discrete-event simulations require state saving only for *excursions* (where an object would predict the future by simulating it). In a Time Warp simulation, on the other hand, frequent state saving is required, in order that objects computing with incorrect values can restart after a rollback.

Time Warp objects communicate by sending messages. All messages have time stamps indicating both the sending and receiving time. If a message arrives in an object's future, the system puts it on the object's input queue for future

[1] This is an expanded version of an article which originally appeared in the April 1990 issue of *Operating Systems Review*

processing. Messages arriving in an object's past cause the object to roll back to a time prior to the message. Messages that have already been sent are cancelled by sending a corresponding *anti-message*. When an object's time is the same as a message's time, the system removes the message from the input queue (also called event queue), evaluates the message text, and processes any output messages. This *object cycle* serves as a basis for our non-preemptive scheduling algorithms. This contrasts with a *preemptive* scheme that might suspend an object's execution in the middle of a cycle. We choose non-preemptive scheduling, because it's difficult to move objects between machines in the middle of an event cycle. Likewise, in our Lisp programming environment, preemptive algorithms require complex and expensive context switches.

Global Virtual Time (GVT) is the time of the farthest behind local clock for all simulation objects. This time is equivalent to the real time of the simulation. All input and output is synchronized with GVT. Since an object cannot send a message into its past, objects can never roll back to a time earlier than GVT. All messages and states saved prior to GVT are subject to *fossil collection*, which frees their storage space. The computation of GVT occurs at periodic intervals.

Since our workstations are quite large (8 - 24 megabyte workstations), each can service many objects. A Time Warp scheduling routine running on each processor optimizes various system parameters to reduce the global program execution time. Because we wanted to scale the system up, we did not allow interprocessor communications between scheduling algorithms. Finally, it is extremely difficult to compute event dependencies from program source code. Consequently, our algorithms assume no known scheduling dependencies.

3 A New Category of Distributed Scheduler

There are many types of single processor schedulers: deadline, priority, preemptive, non-preemptive, etc. Two of the main objectives for all scheduling disciplines are to maximize throughput and reduce overhead [3]. Distributed schedulers must also meet the same requirements. A review of the literature reveals two categories of distributed schedulers.

1. Scheduling of independent tasks

2. Scheduling of dependent tasks

Tannenbaum [11] states that the first type can be scheduled randomly, and then only discusses how to handle the scheduling of dependent tasks. There are also numerous studies of dependent systems [7, 8, 9, 10].

Dependency information is normally not part of the simulation code, nor in general can it be computed automatically. Consequently, the dependency-based schemes are not appropriate for our purposes. Time Warp processes may be scheduled in various orderings and the system will still complete execution.

Thus, Time Warp would appear to fall into Tannebaum's first category. However, our results show random scheduling to be extremely poor. Although there are dependencies in a Time Warp simulation, we can't reliably predict them. Therefore, our schedulers fall into a third category that rely on indirect measures of dependency based on simulation state.

The schedulers discussed in this paper are also not covered by Casavant and Kuhl's[2] taxonomy of distributed schedulers. At the top of their hierarchical structure is *local* versus *global* scheduling. They describe global schedulers as those that decide *where* a task should execute. *Local* schedulers run concurrently (one on each processor) and determine the scheduling for their respective processors only. Since our schedulers make no decision of where a task should execute, they must be local. Casavant and Kuhl discuss subcategories of global schedulers, but their taxonomy stops at this point.

Nonetheless, many of Casavant and Kuhl's subcategories for global schedulers apply to our local schedulers. For instance, some of our schedulers fall into their *dynamic* category, because they use run-time information about the task to determine the scheduling order. They are *noncooperative*, because individual processors make decisions independent of the actions of other processors. Our schedulers also use *sub-optimal approximations* to determine which task to schedule next. For example, Time Warp information such as Local Virtual Time (LVT) and input queue length prove useful for the schedulers but are clearly non-optimal. We will show that the best schedulers are those that are local, non-cooperative, and use sub-optimal approximations. We will also show that a well-chosen scheduling method results in improved Time Warp performance.

4 Time Warp Scheduling Algorithms

Our first task was to identify system characteristics that affect total system throughput: a simulation's start-to-finish wall clock time. We instrumented the Time Warp system to measure various parameters that determine system performance. These include the lengths of Time Warp message queues, standard deviation of LVT across all objects, CPU time per object cycle, number of rollbacks, network utilization, and many others. Starting with the simplier schedulers, we examined advance rates and execution statistics to make inferences on how to improve the GVT advance rate. Though we could not examine all combinations of these system parameters, we choose important combinations that would account for widely varying event times and system load. We tested the following ten scheduling algorithms.

1. **Worst Case**: An unlimited Round-Robin scheduler where each object runs to completion or exhaustion of its input queue. This algorithm works well in simulations with no interaction between objects, but exhibits worst-case and even divergent behavior with dependent objects. We were not able to test this scheme on any but the most trivial simulations.

2. **Round-Robin**: We implemented this algorithm as a control in the experiment. This is essentially the random scheduling algorithm in Tannenbaum's taxonomy. Each object executes a fixed number of object-cycles before relinquishing control. A very large number of object-cycles would make the Round-Robin scheduler degenerate into the Worst Case scheduler. By experiment, we've determined that three is the best number of cycles.

3. **Lowest LVT First**: This algorithm isolates LVT as a measure for scheduling. The idea is to process the object with the lowest LVT first to get it closer in time to other objects in the system. This scheduler is essentially the Round-Robin scheduler in which the order of object execution is changed, but the number of cycles per object remains constant. Each time the system cycles through the queue, it places the object with the lowest LVT at the head. The reasoning behind this algorithm is as follows: rollback occurs when object **B** sends a message, **X**, to object **A** with a time stamp earlier than **A**'s current LVT. Thus, object **A** must roll back to a time previous to **X**'s time stamp.

4. **Priority LVT**: Like the Lowest LVT First algorithm, this scheduler emphasizes objects with low LVTs. It computes the standard deviation of the LVTs of the processor's objects. It then places all objects with LVTs less than the mean LVT plus the standard deviation onto a priority scheduling queue. Objects on the priority queue get more event cycles than objects on the standard scheduling queue. The idea of both this and the Lowest LVT scheme is to allow objects that are farthest behind to catch up with the others.

5. **Largest Queue Priority**: This algorithm emphasizes objects with large unprocessed event queues. It is exactly the same as Priority LVT except that it computes the standard deviation of unprocessed event queue sizes. When an object's queue length is greater than the mean plus the standard deviation, it goes onto the priority queue. The belief is that a large event queue predicts a large processing requirement. Processing more of the events should even out the wide disparity in LVTs, resulting in improved performance.

6. **Queue Cycle**: This algorithm emphasizes queue length rather than LVT as a heuristic for measure of performance. This algorithm simply schedules the object with the largest unprocessed event queue next. It schedules each object for three cycles.

7. **Bradford/Fitch**: The Bradford/Fitch algorithm attempts to emphasize objects doing useful work by penalizing objects with high LVTs. For each object, it computes a penalty value. Out of those objects with the lowest penalty, it picks one at random and cycles it up to three times. After

each object cycle, the Bradford/Fitch algorithm checks if any rollbacks or returned messages occurred since it last checked. A returned message indicates that an object's input queue is full and is thus returning any new ones to the sender. Rollbacks and returned messages suggest that other objects need to be processed for the system to continue doing useful work. Thus, if any are detected, the scheduler then finds the object with the lowest LVT and schedules it; otherwise, the same object is rescheduled.

8. **Anti-Penalty**: This algorithm is a variation on the Bradford/Fitch scheduler that attempts to stop the cycling for anti-messages and anti-reminders. Anti-reminders are generated by the priority request mechanism [5]. Like the Bradford/Fitch scheduler, the penalty function finds the object with the lowest LVT to schedule. However, it continues scheduling that object for up to three cycles or until a rollback, returned message, anti-message, or anti-reminder shows up. Bradford/Fitch only considers returned messages and rollbacks.

9. **Queue Anti-Penalty**: This variant of the Anti-Penalty and Bradford/Fitch algorithm uses unprocessed event queue length, rather than lowest LVT, as the penalty function.

10. **Positive Infinity**: This algorithm puts objects that have reached positive infinity, meaning they have nothing more to do, on a low priority queue. It schedules objects below positive infinity three times more often than those at positive infinity, because objects at positive infinity have finished working and are just waiting for the system to complete or to roll back. This scheduler focuses on the termination condition in Time Warp, while ignoring the interval previous to termination.

There are a large number of system parameters that are potential measures of system performance. We selected ones that seemed reasonable based on our understanding and experience with the Time Warp system. Attempting to cover all possible combinations of these measurements is beyond the scope of this paper. However, we have tested a reasonably wide range and have shown that certain combined measurements are better indicators of system performance than others.

5 A Test Simulation

Because of time constraints, we picked a single worst case Time Warp simulation in which all objects communicate with all other objects at every simulation time step. We maintained constant parameters for the execution of the simulation varying only processor load and scheduling algorithms.

The simulation consists of a number of moving objects which have pre-assigned velocities. These move about on a grid and change direction when

they bump into the surrounding walls or when two or more land on the same grid. A graphics object displays moving object location on a regular basis, and a console object prints the numeric location and direction changes. All objects were programmed to accept information in a non-deterministic order. However, with the exception of the console display, the simulation is still deterministic.

The simulation was run on a network of four SUN-4 workstations connected by a ten megabit local area network. Each processor has its own local disk to avoid paging over the network. The network was run without any load other than that created by Time Warp. The controlling workstation is a 10 MIP processor with 24 megabytes of main memory running the 12 megabyte GVT and graphics process. The three Time Warp processors are eight megabyte SUN-4's running a 12 megabyte process with three objects per machine. There is no dynamic load balancing (dynamic load balancing is the subject of another paper [1].

Measured wall-clock time does not include Time Warp system initialization. However, the times are effected by the size of the Time Warp GVT interval as the last GVT cycle may have few events. With a GVT cycle time of five seconds and a minimum simulation execution time of 100 seconds, there is an error of only five percent.

6 Experimental Results

We tested the ten algorithms described in the previous section. The worst-case scheduler is more than $O(n^3)$ where n is the number of messages between objects. Only in extremely simple cases did it ever complete and is therefore excluded from the figures.

To compare the performance of the algorithms, the test simulation was run with each 16 times. As we are running a public network of workstations, network delays and uncontrolled processor loads cause a variance in execution time. A good algorithm will also be responsive to these independent changes. The histograms in Figures 1-9 show the distribution of execution times for the different schedulers. For example, Figure 1 shows that the Lowest LVT First scheduler executed nine times at 110 seconds, six times at 120 seconds, and one time at 130 seconds.

The Anti-Penalty scheduler produced the best performance, followed by a tie between the Queue Anti-Penalty scheduler and the Bradford/Fitch scheduler. The results show the correctness of the assumptions made by the Bradford/Fitch scheduler: scheduling the object with the lowest LVT next and blocking when it can make no more progress reduces simulation wall-clock time.

The only difference between the Anti-Penalty scheduler and the Bradford/Fitch scheduler is when the scheduler blocks the currently executing object. The Bradford/Fitch scheduler blocks as soon as a rollback or a returned message is sent, because any more work done by the currently executing object at this point will

be wasted. The Anti-Penalty scheduler goes one step further. It also blocks an object if anti-reminders or anti-messages are sent. Anti-messages indicate that an object will be rolled back, so it is best to block the current object so that the object to roll back can do so immediately, because any messages it gets after the anti-message are likely to be undone. It also considers anti-reminders, because the priority request mechanism can cause an object to roll back. The results show that blocking on anti-messages and anti-reminders in addition to rollback or returned messages enhances the performance of the scheduling algorithm.

These results show that LVT, anti-messages, rollbacks, returned messages, and anti-reminders are the best indicators of system load. Input queue size is also an important factor, but when taken with or without LVT, it does not produce results as good as using LVT alone. The Round-Robin scheduler was one of the worst performers. The poor performance of the simple Round-Robin scheduler indicates the advantages of using state information to determine the scheduling order in the Time Warp system.

Figure 10 ranks scheduler performance, with the best at the bottom. For each scheduler, there is a bar indicating the range of times it took to execute. The shaded portion of the bar indicates times below the mean, and the unshaded portion of the bar illustrates the times above the mean. The mean time for each scheduler is the dividing line between the shaded and unshaded regions.

From Figure 10, we can see that there were three groups of schedulers: those whose times ranged from 100 to 110, from 100 to 130, and from 100 to 200. The three schedulers in the fastest group all depend on aspects of the object state information. This shows the dramatic difference the use of Time Warp state information in scheduling decisions can make on system performance.

7 Conclusions

We tested ten different algorithms for the Time Warp system. Through timing comparisons, we have shown that the best performance is achieved by examining information about the object's state to determine which object to schedule next. These algorithms fall into a new category for distributed system schedulers. The previous categories consisted of schedulers for independent processes and schedulers for processes with nested dependencies. The schedulers have attributes of Casavant and Kuhl's taxonomy: they are local, noncooperative, and use sub-optimal approximations, but they don't firmly fit into that taxonomy either. A category that applies to these schedulers would cover local schedulers of independent processes that benefit from the use of process state information in the scheduling decision.

Benchmarks of the schedulers showed that the Anti-Penalty scheduler performed better than the others. The Anti-Penalty scheduler selects the object with the lowest LVT to schedule. It continues scheduling it for up to three cycles or until a rollback, returned message, anti-message, or anti-reminder shows up.

The benchmarks were executed under a simulation executed on a five-processor Time Warp system. The benchmarks also showed that LVT, anti-messages, roll-backs, returned messages, and anti-reminders are the best indicators of system load on Time Warp.

The dynamic load balancing techniques that we have tested use the same inputs for measuring processor load under Time Warp as do the scheduling algorithms. While a description of load balancing is left to another paper, it is important to point out that the same parameters for measuring processor load apply to both scheduling and load balancing.

References

[1] C. Burdorf and J. Marti. Load balancing strategies for time warp on multi-user workstations. In preparation.

[2] Thomas L. Casavant and Jon G. Kuhl. A Formal Model of Distributed Decision-Making and its Application to Distributed Load Balancing. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, 1986.

[3] H. Deitel. *Introduction to Operating Systems.* Addison-Wesley, Reading, Mass., 1984.

[4] J. P. Fitch. A Loosely Coupled Parallel LISP Execution System. In *The Design and Application of Parallel Digital Processors*, 1988.

[5] Barbara Gates and Jed Marti. An Empirical Study of Time Warp Request Mechanisms. In *Distributed Simulation*, number 3, San Diego, California, 1988. Simulation Councils, Inc.

[6] D. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3), July 1985.

[7] Errol Lloyd. Two Processor Scheduling with Limited Preemption. *Performance Evaluation*, 6:307–312, 1986.

[8] S. Lo and V. D. Gilgor. Properties of Multiprocessor Scheduling Algorithms. In *Proceedings of the 1987 International Conference on Parallel Processing*, 1987.

[9] R. Mehrotra and S. Talukdar. Scheduling of Tasks for Distributed Processors. In *Proceedings of the Eleventh International Symposium on Computer Architecture Conference Proceedings*, pages 263–270, 1984.

[10] Sartaj Sahni. Scheduling Multipipeline and Multiprocessor Computers. *IEEE Transactions on Computers*, 33(7):637–645, 1984.

[11] A. S. Tannenbaum. Distributed Operating Systems. *Computing Surveys*, 17:419–470, Dec. 1984.

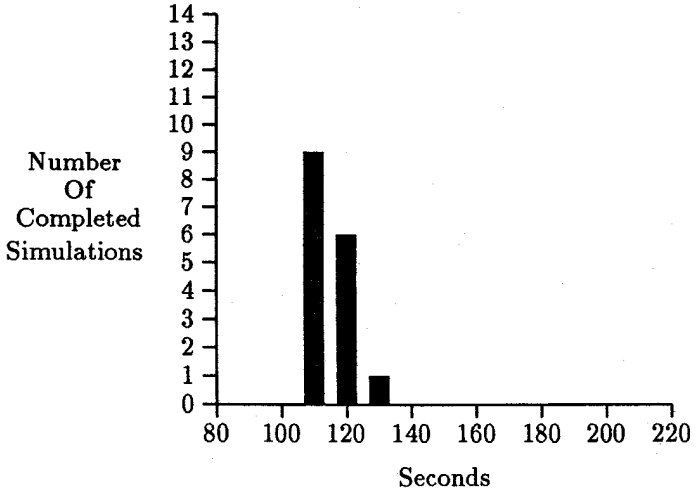

Figure 1: Lowest LVT First Scheduler

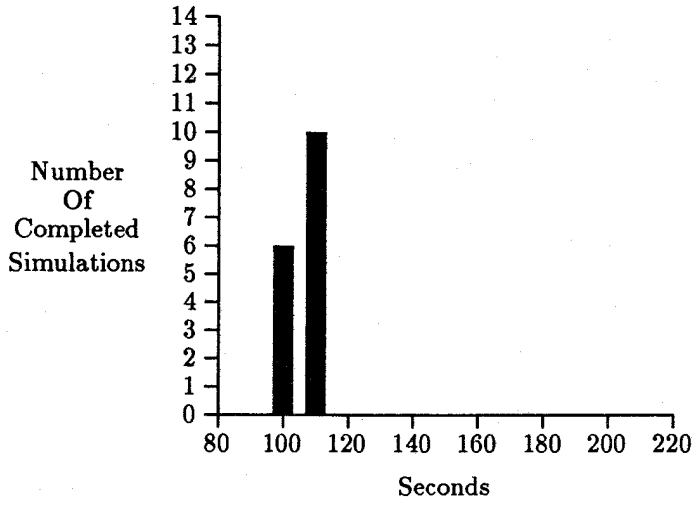

Figure 2: Anti-Penalty Scheduler

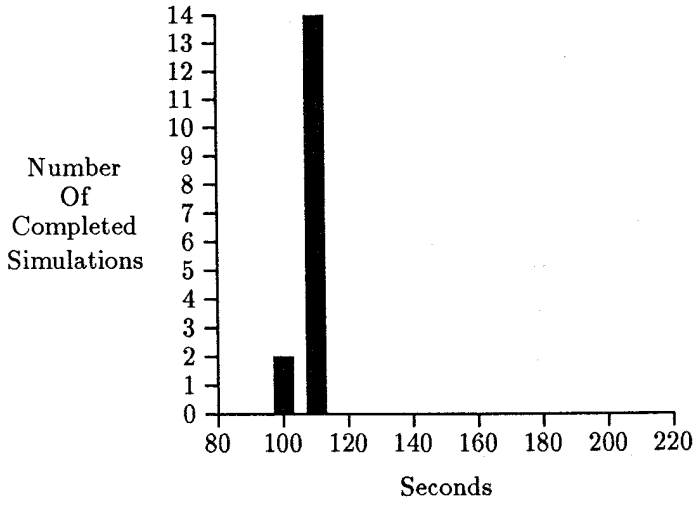

Figure 3: Queue Anti-Penalty Scheduler

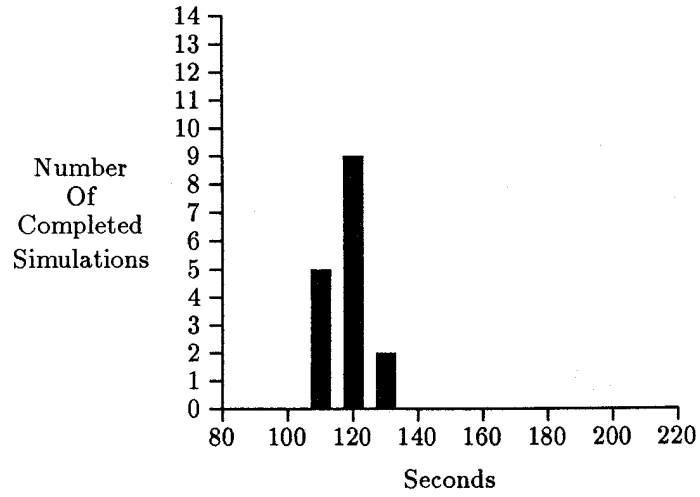

Figure 4: Queue Cycle Scheduler

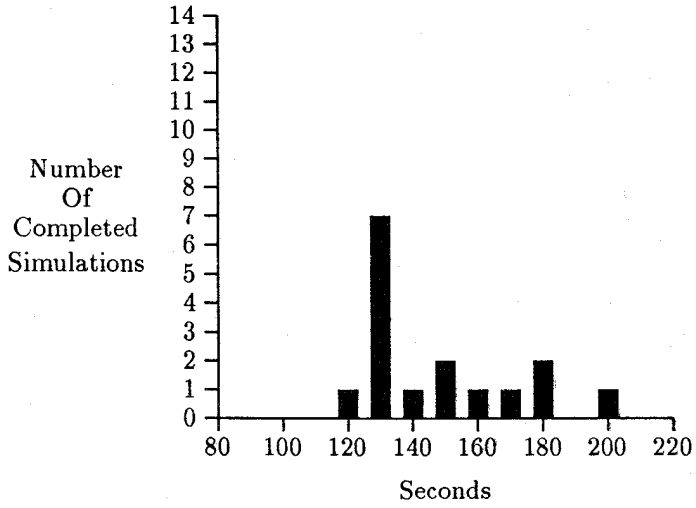

Figure 5: Positive Infinity Scheduler

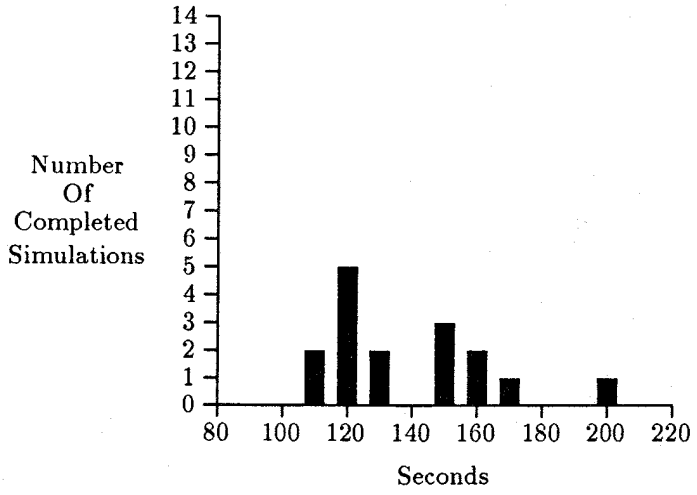

Figure 6: Round-Robin Scheduler

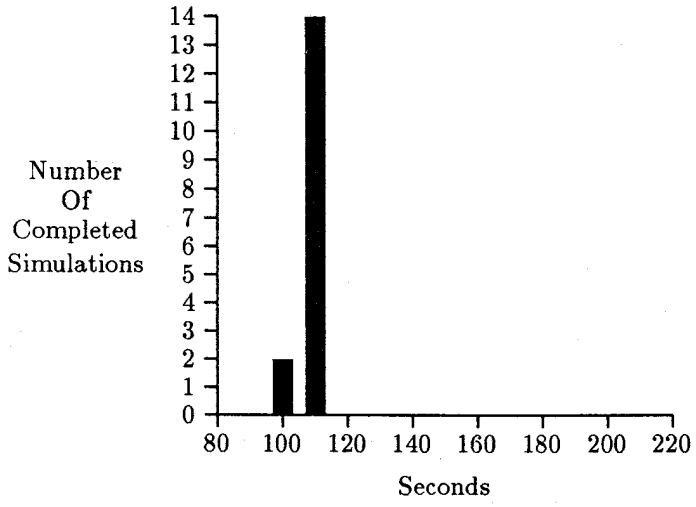

Figure 7: Bradford/Fitch Scheduler

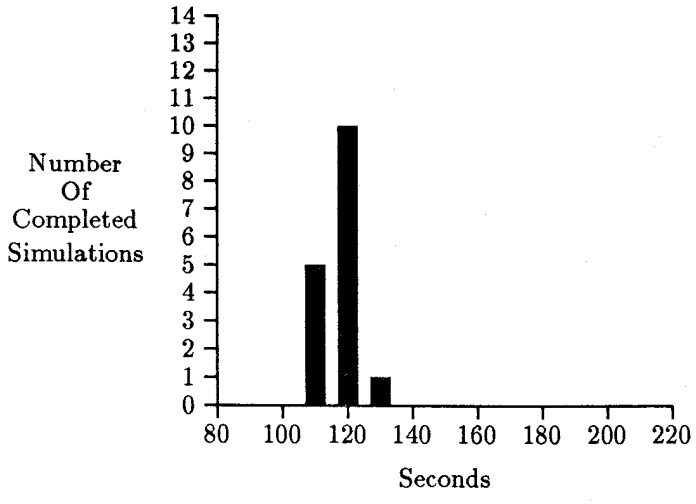

Figure 8: Priority LVT Scheduler

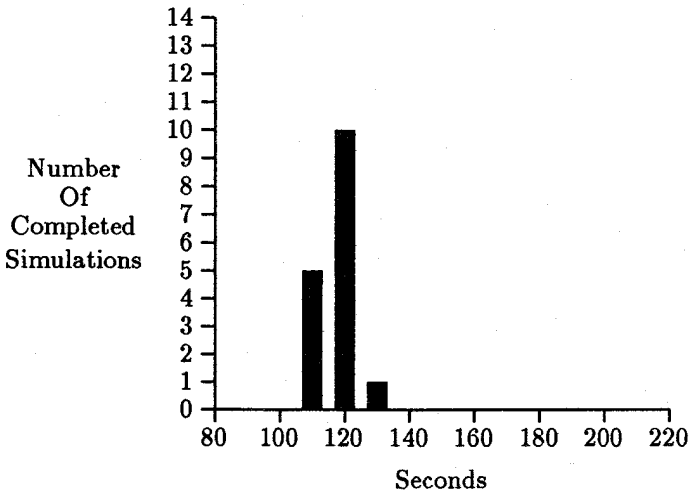

Figure 9: Largest Queue Priority Scheduler

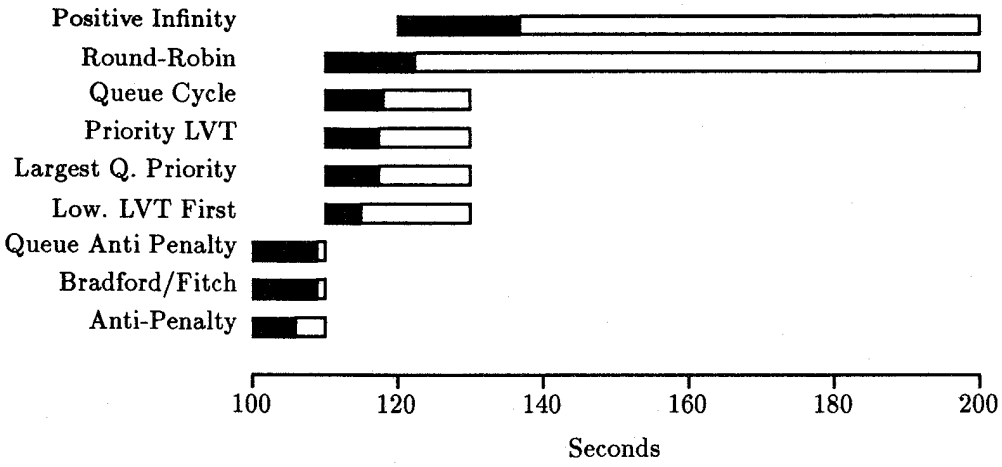

Figure 10: Summary of Schedulers