5-2006

Task Scheduling and Simulation

Martin Lee Mayne
Utah State University

TASK SCHEDULING AND SIMULATION

by

Martin Lee Mayne

Thesis submitted in partial fulfillment of the requirements for the degree

of

HONORS IN UNIVERSITY STUDIES WITH DEPARTMENT HONORS

in

Computer Science

Approved:

Thesis/Project Advisor

Dr. Xiaojun Qi

Department Honors Advisor

Myra Cook

Director of Honors Program

Dr. Christie Fox

UTAH STATE UNIVERSITY
Logan, UT

2006

1. Introduction

Scheduling is a problem that is not only common computer science, but which also comes up in a variety of real world situations. Whether multi-threading computer processes, scheduling airport traffic, optimizing assembly line production and manufacturing, or ensuring that enough employees are on the floor at a given time, scheduling is important for increasing efficiency, reducing costs, optimizing production, and meeting deadlines. Scheduling has been used throughout history and has increased in efficacy through the years. Modern computerized scheduling techniques are typically much better than human calculations and often produce surprising insights and results which would have been otherwise passed over due to their counter-intuitive nature [6].

The techniques for solving scheduling algorithms range from simple and intuitive to complex and computationally intensive and they vary in efficiency. The most obvious scheduling technique (or lack thereof, perhaps) is the first-come-first-serve (FCFS) scheduling algorithm. This algorithm simply places tasks in a queue and handles them in the order they became known. We see the first-come-first-served approach frequently in real life: checking materials out of a library, lunch lines, printing on a shared printer, free sample giveaways, etc. This approach can be satisfactory for simple or independent tasks. Sometimes, in real life, it is even expected and can be a source of aggravation when not followed (e.g. line-jumpers) [4]. However, in situations involving prioritized tasks or precedence constraints (task B cannot begin until task A is finished), first-come-first-served quickly becomes unsatisfactory.

In such circumstances, a more sophisticated scheduling scheme is needed. In the 1950's when the first nuclear-powered submarine was constructed, a better scheme known as "the critical path method" was devised [7]. The critical path (CP) method is not actually a scheduling algorithm, but rather a tool on which to base a scheduling algorithm. Many successful modern algorithms are based on the critical path method. The critical method works by first doing a forward pass through the task graph to establish the earliest possible start time for each task, and then making a backward pass to establish the latest possible start time for each task. Tasks that have equal earliest and latest start times are called "critical tasks" and are given highest priority in the schedule. Other tasks are "floating" and can be delayed up until their latest starting times without affecting the overall completion time of the project [8].

2. Arithmetic Implementation

For my project, I implemented both the FCFS and CP methods. There is no pseudo code for the first-come, first-served algorithm: the first available task is simply run to completion and then the next available one is scheduled [2]. Note that tasks are subject to precedence constraints. This means that a task is not considered "available" until all of its predecessor tasks have been scheduled. Thus, starting with the "root tasks," each task is scheduled by placing it in a queue as it becomes available (i.e. once all its predecessors are scheduled).

Critical path scheduling is done in three phases, the first of which is essentially the FCFS method. Chris Hendrickson [3] gives the pseudo code for the critical path is as follows:

- Event numbering:
    1. Give the starting event number 0.
    2. Give the next number to any unnumbered event whose predecessor events are already numbered.

3. Repeat step 2 until all events are numbered

- Earliest Event Time Algorithms:
  1. Let E(0) = 0.
  2. For j = 1, 2, 3, ..., n (where n is the last event), let maximum E(j) = maximum{E(i) + D_{ij}} where the maximum is computed over all activities (i,j) that have j as the ending event.
- Latest Time Algorithms:
  1. Let L(n) equal the required completion time of the project. Note: L(n) must equal or exceed E(n).
  2. For i = n-1, n-2, ..., 0, let L(i) = minimum{L(j) – D_{ij}} where the minimum is computed over all activities (i,j) that have i as the starting event.

Basically, the algorithm propagates from the beginning to find the earliest start times and then propagates backwards to find the latest start times. Once the critical path has been calculated, the schedule can be created. The most straightforward CP scheduling algorithm is simply to give critical tasks priority over other available tasks. If two critical tasks have equal critical times either task can be scheduled first. Obviously, the precedence constraints of the task graph must be satisfied before critical times are considered [1].

3. Simulation

3.1 Simulation Tool

Dr. Qi and I chose to implement the project using XJ Technologies' AnyLogic software [9]. AnyLogic is a relatively new general purpose simulation package which has been used to simulate and solve problems related to pedestrian dynamics, ecosystems, effects of marketing on product adoption, and more. AnyLogic's flexibility, Java-based platform, and visual tools make it especially nice to work with. To our knowledge, it has never been used for scheduling and simulating task graph systems.

3.2 Graphical, Drag-and-drop Task Graphs

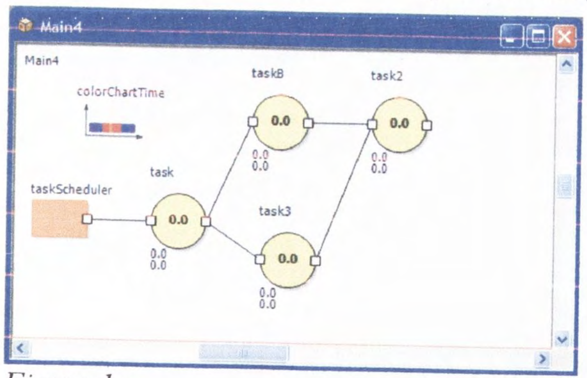


Figure 1a.

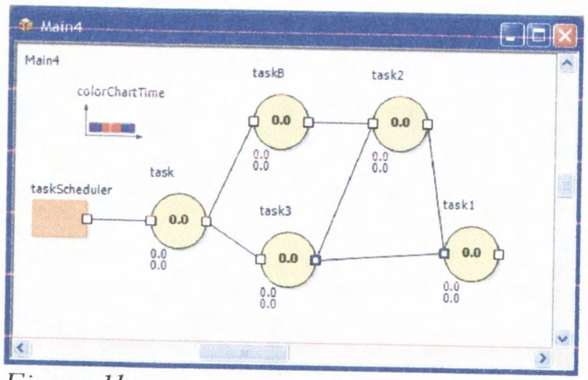
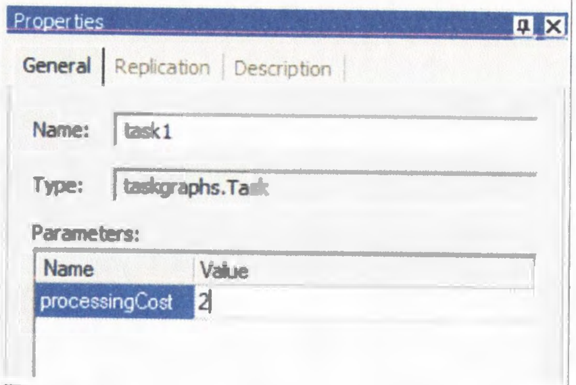


Figure 1b.

AnyLogic's environment is especially nice for its drag-and-drop graphical interface. The first step of my project was to create the task graph system consisting of tasks and a scheduler. This system allows a user to create arbitrary task graphs by dragging tasks onto the workspace and connecting them together, as shown in Figure 1. The task graph reads from left to right, so that tasks connected to the left side of another task are its predecessors and tasks connected to the right side of a task are its successors.

The scheduler connects to the "root tasks" from the left.

In addition to the precedence relation that is established by the graph (how the tasks are connected together), each task is also given a cost by clicking on it and setting its properties. This cost is generally associated with time. The scheduler allows the user to select a scheduling algorithm of his or her choice (from those available). Figure 3 demonstrates how to changed the properties of a task.


Properties

General | Replication | Description

Name: task1

Type: taskgraphs.Task

Parameters:

Name	Value
processingCost	2

Figure 3.

Additionally, I experimented with graphing the results on a Gantt (or time) chart. Gantt charts are a common way of displaying schedules. Tasks are displayed as horizontal bars with the left side of the bar marking the time the task begins execution and the right side marking the time its execution ends. Usually, tasks will be given different colors to distinguish them from each other on the chart. To simplify the implementation, I simply alternate between two colors to indicate when one task has ended and another begun.

Finally, AnyLogic has the ability to animate the model while it executes. I took advantage of this capability by having the tasks change color to indicate their state – whether waiting, ready, executing, or completed. The earliest start times, latest start times, and costs are also displayed on each task.

3.3 Scheduling

The scheduling is done in three phases:

1. Initialization

At start time, each task sends a message to its predecessors and consequently receives a message from all of its successors. Each task keeps a list of its successors and the scheduler makes a list of the root tasks.

Next, the scheduler sends a message to the root tasks. The root tasks forward the scheduler information to their successors. This step not only allows each task the ability to reference the scheduler, but also gives them the opportunity to make a list of their predecessors.

At this point, the scheduler and tasks are all connected together (internally) in a dynamic graph and the scheduling begins..

2. Scheduling

The scheduler schedules the tasks (puts them in a queue) according to the algorithm selected.

3. Execution

The scheduler executes the schedule and the results are shown in real time.

4. Experimental Results

To demonstrate the experimental results, I simulated the example model from the earlier figures using both the FCFS-based and CP-based scheduling schemes.

Figure 4 shows the FCFS-based scheduling. It shows an executing task graph system. The graph itself shows that the left-most and upper left tasks have already been completed and that the bottom middle task is running, shown in gray and green respectively. The Gantt chart at the bottom of the window also shows that two tasks have run and that they each took one second. The FCFS algorithm is running – notice how the earliest and latest starting times are displayed as ∞ and -∞ because they are ignored and never calculated.

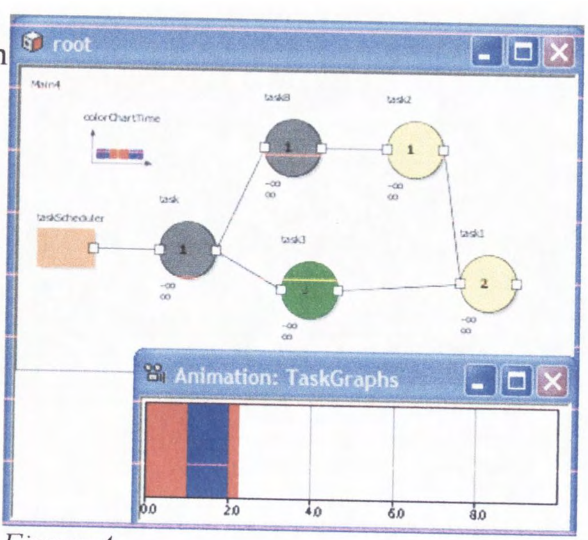


Figure 4.

Figure 5 shows the CP-based scheduling and the earliest and latest start times have been calculated. This task graph only has one non-critical task: the green one that is running. Notice how, using the critical path, the bottom middle task runs before the upper left task because it is a critical task. Critical tasks have equal earliest and latest starting times as described earlier, and the system emphasizes them by outlining them with a darker line. Also, the Gantt chart shows that the second task executed took three times longer to complete than the first task.

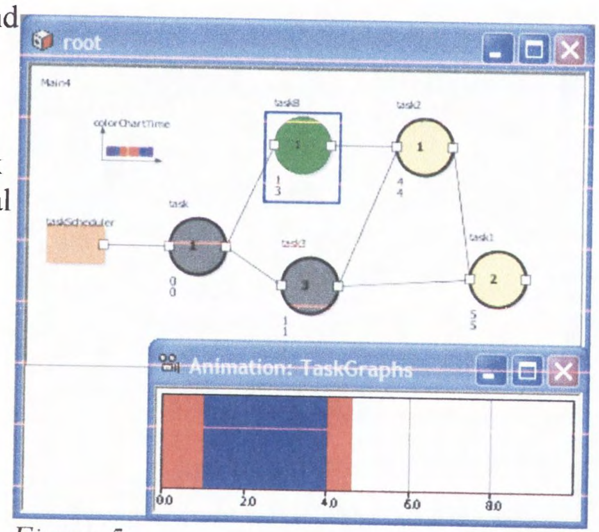


Figure 5.

5. Conclusion

First-come-first-served and critical methods or just the beginning of scheduling algorithms. The next step would be to incorporate multi-processor scheduling. Eventually, this system could be expanded to include a wide array of scheduling algorithms over a large range of task graph scheduling domains. With some luck, the system could eventually be used to solve real-world problems. At any rate, the project gave me experience with scheduling algorithms, the AnyLogic software package, and the Java programming language.

6. References

[1] Bowen, Larry. "Scheduling Algorithms." The Mathematics of Scheduling: Directed Graphs and Critical Paths. University of Alabama. Accessed May 2006. <http://www.ctl.ua.edu/math103/scheduling/scheduling_algorithms.htm#Critical-Path%20Algorithm>.

[2] Callari, Franco. "First come, first served." Operating Systems Class Home Page. 1996. May 2006. <http://www.cim.mcgill.ca/~franco/OpSys-304-427/lecture-notes/node41.html>.

[3] Hendrickson, Chris. "Fundamental Scheduling Procedures." Fundamental Concepts for Owners, Engineers, Architects and Builders. Ch 10. Pittsburgh: 1998. May 2006. <http://www.ce.cmu.edu/pmbook/>.

[4] Larsen, Richard C. Perspectives on Queues: Social Justice and the Psychology of Queueing. Operations Research Society of America. 1987. May 2006. <http://www.jstor.org/view/0030364x/ap010191/01a00130/0>.

[5] "Scheduling (computing)." Wikipedia, The Free Encyclopedia. 4 May 2006, 02:39 UTC. May 2006. <http://en.wikipedia.org/w/index.php?title=Scheduling_%28computing%29&oldid=51466846>.

[6] "Scheduling (production processes)." Wikipedia, The Free Encyclopedia. 2 May 2006, 10:35 UTC. May 2006, <http://en.wikipedia.org/w/index.php?title=Scheduling_%28production_processes%29&oldid=51190349>.

[7] U.S. Army, Corps of Engineers, Construction Engineering Research Laboratories (CERL). "History." CPM Tutor. 13 Dec 1996. May 2006. <http://www.buildersnet.org/cpmtutor/html/history.html>.

[8] U.S. Army, Corps of Engineers, Construction Engineering Research Laboratories (CERL). "Scheduling." CPM Tutor. 13 Dec 1996. May 2006. <http://www.buildersnet.org/cpmtutor/html/scheduling.html>.

[9] XJ Technologies Company. AnyLogic. 2005. <http://www.xjtek.com/anylogic/>.