A Comparison of Scheduling Algorithms

Samuel Jackson
Aberystwyth University
slj11@aber.ac.uk

ABSTRACT

This document aims to analyse four different scheduling algorithms (namely Shortest Time Remaining, First Come, First Served, Round Robin and Lottery Scheduling) to compare and contrast their performance in a simulated environment and evaluate their strengths and weaknesses. This was done by creating a set of experimental test files containing jobs exhibiting a variety of features likely to be present in a real world scheduling environment and running these on the simulator. The results showed a clear trend towards time slice based schedulers having a higher mean job duration but a low idle time. The reverse appeared to be exhibited by queue based approaches. By analysing the results of these tests it was possible to determine the major differences between the four approaches where a clear trade off between throughput and mean job duration and context switches versus idle time and the possibility of starvation occurred.

Categories and Subject Descriptors

D.4 [**Operating Systems**]: Process Management—*Scheduling*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Algorithms, Performance, Measurement

Keywords

Algorithms, Scheduling, Performance, Lottery Scheduling, Shortest Time Remaining, Round Robin

1. INTRODUCTION

This report aims to examine the differences between three approaches to scheduling algorithms that are widely used in systems throughout computer science where multiple tasks require computation in parallel. Scheduling algorithms are used to organise and prioritise jobs of computation according to a set scheme which deals with the issues such as throughput, turnaround and context switches.

The scheduling algorithms used here are First Come, First Served (FCFS), Shortest Time Remaining (STR), Round Robin and Lottery scheduling which were chosen because they all exhibit characteristics which make them suitable in different situations. The former three are deterministic in nature while lottery scheduling is non deterministic. FCFS will work on the first job given to the scheduler until it finishes or is blocked for I/O. I which case it will be returned to the back of the queue. STR takes a simple priority queue based approach where jobs are ordered according to the amount of time left before they finish processing. Round Robin uses equal time slices to share processing time between every job in the queue. Finally, Lottery scheduling uses probabilistic methods to give higher priority jobs a greater likelihood of being selected. In the implementation use in this paper Lottery scheduling treats lower priorities as being of greater importance.

2. METHODOLOGY

To thoroughly test the performance of the algorithms, each of the schedulers was exercised using a set of data files each of which test different properties important to scheduling algorithms. This includes files which contain a heavy amount of I/O processing, a low level of I/O processing, jobs where all priorities are the same, and some general test data. Test data that is heavy in I/O processes exercises how well an algorithm handles a high level of context switching due to jobs being blocked for reading or writing. The data file with a low level of I/O is used to evaluate the performance of the scheduler in situations which would not require I/O. A file where all priorities are the same is used to help evaluate lottery scheduling (which schedules according to priority) to evaluate its worst case performance.

Th method used to test each algorithm was to run it with each of the supplied generic data files and to also test it with the custom data files to evaluate how each algorithm handles specific aspects of scheduling. The results are then compared against one another using five attributes important to scheduling algorithms (mean duration, total CPU time, number of context switches, idle time and time for first job to finish) in order to show how each algorithm performed both with generic test data and specific test data to gain an idea of how well each performed both generally and within a specific area of performance.

Because lottery scheduling is non-deterministic it required a different approach to testing that involved an extra step.

Lottery scheduling gives different results on each run, therefore in order to evaluate an average run an automated simulator (code included with alongside the schedulers) was created to simulate 10,000 runs of the algorithm per test file. The results of this data could then be averaged out to representatively compare lottery scheduling's general case performance with the performance of the other algorithms. It is this average which has been used in comparisons with the other schedulers in the following section and it is therefore important to remember that the value generated by lottery scheduling will vary, but will on average behave like the value given.

3. RESULTS

This section details the outcomes of testing each of the algorithms using the data files outlined in section 2 to measure four key attributes important to scheduling algorithms. As mentioned in the preceding section, the values used with lottery scheduling are the mean results of 10,000 tests.

3.1 Mean Duration

Mean duration is the average amount of time that a scheduled job takes to finish processing. Figure 1 shows that the mean duration time across all tests and algorithms was fairly consistent, with Shortest Time Remaining (STR) having the lowest mean duration in all tests. Round Robin scheduling showed the worst performance of the four with means durations slight above those of FCFS and Lottery scheduling, both of whom presented mixed results depending on the test.
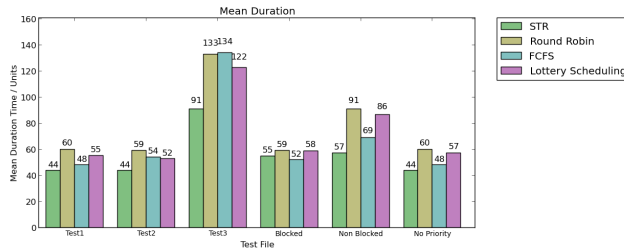

Figure 1: Mean Job Duration across all tests and algorithms

The mean duration appears to spike on test 3 and on the test with no I/O blocking, although STR appeared to show a significantly lower mean duration time for the no I/O test than the other schedulers.

3.2 CPU Time

CPU time is the number of CPU ticks that each algorithm performs in total in order to complete all of the jobs. Figure 2 shows that the CPU times were also fairly consistent across all tests with Round Robin besting or equalling the other algorithms. Conversely, STR performed slightly worse in this criteria than the others. Also, FCFS exhibited a higher CPU time at 191 ticks as opposed to a consistent 182 ticks for every other algorithm.
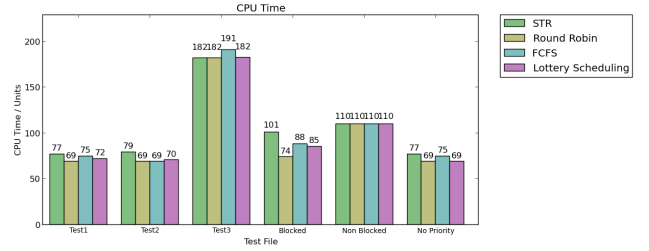

Figure 2: CPU time to complete processing jobs across all tests and algorithms.

3.3 Context Switches

Context switches measure the number of times the algorithm switches between different jobs while executing them (if at all). Figure 3 shows a mixed result between FCFS and STR in terms of the number context switches made with FCFS having the lowest result on tests 1,2, the high I/O blocking test and the test with all priorities the same and STR scoring lower on the other two tests. Round Robin had the worse performance overall with Lottery scheduling following closely behind.

Interestingly, FCFS had a very high number of context switches on tests 3 and the non I/O blocking test when compared with its closest counterpart STR. In fact, the STR algorithm exhibited the lowest number of context switches in the non I/O blocking test. This is in sharp contrast with it giving the second highest number of context switches in all other algorithms.
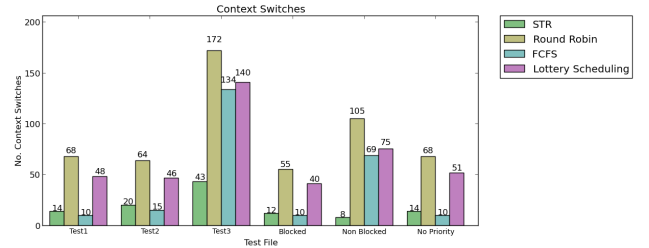

Figure 3: Total number of context switches across all tests and algorithms.

3.4 Idle Time

Idle time is the number of CPU ticks that occur where there were no jobs pending within the scheduler as they were all either finished on blocked for I/O. Figure 4 shows the total idle times for all tests as a percentage of the CPU time which clearly shows Round Robin to be the winner with only 5 idle cycles during the high I/O blocking test and 0 idle time for every other test. Lottery scheduling performed reasonably we, while STR and FCFS showed mixed results. Interestingly, STR had no idle time in test 3 while FCFS had no idle time in test 2. STR had a significantly higher idle time during the non blocking I/O test with 31.7% of its CPU time spent idle. The high I/O blocking test proved to show the largest impact on the idle time compared with the other tests.
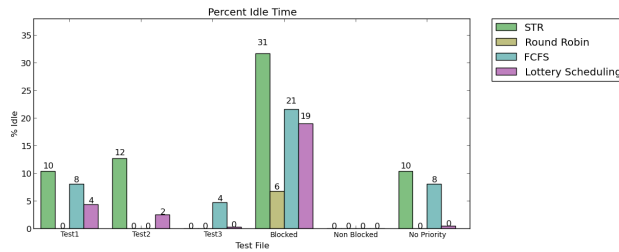
Figure 4: Total idle time as a percentage of CPU time across all tests and algorithms.

3.5 First Job Finish Time

This final section shows the time taken for the first job to finish across all tests. Figure 5 clearly shows that STR to be significantly faster than both Round Robin and Lottery scheduling as well as being generally faster than FCFS. Round Robin has the slowest performance in this category, with test 3 taking 92 cycles in order to finish the first job compared to just 15 cycles using STR.
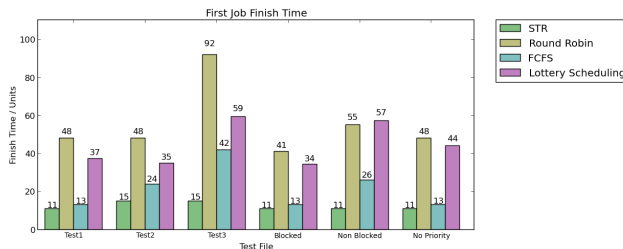
Figure 5: Time for the first job to finish across all tests and algorithms.

4. DISCUSSION

This section discusses the results presented in section 3 to examine the strengths and weaknesses of each algorithm in each of the following categories: first job finish time, mean job finish time, and overall processing time.

4.1 First Job Finish Time

The results shown in subsection 3.5 clearly suggest that the Shortest Time Remaining algorithm is the fastest at finishing the first job. This seems logical because STR priorities new and returning jobs by the amount of processing that they ave left and therefore the STR will put the shortest jobs towards the start of the queue and longer jobs towards the back. This means that regardless of the test data, SJN puts the shortest job at the front and this job will get immediate processing and (providing it does require an I/O block, will finish in the shortest possible amount of time possible.

This is in contrast with the other three algorithms which do not order their jobs according to time remaining. Round Robin scheduling had the worst result in this category. Again, this seems logical because Robin Robin allocates equal CPU time slices to to each job regardless of its size or priority, suggesting that the shortest job will take much longer to finish as processing time must be shared equally between this job and the other jobs in the queue. The is confirmed

by the data shown in figure 5 where the first job in test 3 (by far the longest set of jobs) takes much more time to finish than in any other test. The slightly higher result for the test with no I/O blocking shows a higher finishing time for the first job. This is presumably because the less jobs that get blocked, the more jobs are in the queue and the longer a job must wait for more CPU time.

The first come, first served algorithm appeared to be the second best choice after STR with two thirds of the first job tests finishing with 10 cycles of STR. However, as with Round Robin, FCFS does not prioritises the shortest job, but rather accepts the first job added to the queue and and continues working on it until it finishes or if it becomes blocked for I/O, in which case it is returned to the back of the queue. Therefore the time taken for the first job to finish in FCFS depends upon two factors which are the size of first job in the queue and whether the job will become blocked for I/O.

If the job becomes blocked for I/O then it will be added to the back of the queue and will have to wait for all of the processes in front of it to finish before it will get CPU time again. If there are lots of jobs that require being blocked for I/O (such as those shown in the high I/O blocking test) then FCFS will begin to cycle through jobs rather than concentrating on a single job until it's done.

Lottery scheduling appeared to come off as the average case in the results presented in subsection 3.5. As with Round Robin and FCFS, the implementation of lottery scheduling used here does not prioritise jobs according to their length (although it could be implemented to prioritise by job length) but instead uses the priority assigned to be job inside the testing file (which is presumably a arbitrary number according to the current importance of the job on the system). Because jobs are weighted according to priority and not the size of the job, and because this implementation is pre-emptive, there is no guarantee about the length of time taken for the first job to finish. The speed that the initial job will finish in is decided by a jobs priority and random chance.

4.2 Mean Job Duration Time

When considering the mean time to finish a job, It appears that the Shortest Time Remaining algorithm proves to be the best option. Referring to Figure 1 we can see that on average STR had the shortest mean time to finish a job compared with any other algorithm. This would appear to coincide with expectations as the STR algorithm always attempts to bring the shortest job to the front of the queue which mean they finish very quickly thus pulling the mean duration down. However, this comes with the caveat that longer processes can be starved of CPU time when there are lots of small jobs. This could be a particularly nasty problem if there is a constant stream of small jobs being added to the queue as this would force the mean duration to grow much higher as longer jobs would have to wait a long time.

The second best performing algorithm in this section was First Come, First Served which exhibited only slightly higher mean durations across all tests except in the case of test 3 where is was the worst performer and test 2 where it was nar-

rowly beaten by lottery scheduling. Once again this seems to correlate with predictions. FCFS processes jobs as they arrive, so a queue with only a few short jobs will perform fairly similarly to STR. However, when are a lot of jobs on the queue the algorithms performance begins to deteriorate as longer processes begin to hold the CPU meaning shorter or higher priority jobs get starved of resources. Also, because of the lack of prioritisation, jobs that have been blocked are returned to be back of the queue regardless of the amount of processing left which means nearly complete tasks can be placed right at the back of the queue increasing their duration.

As for the other two tests, figure 1 shows that Lottery scheduling came a narrow third with Round Robin coming in last. The reason for this is most likely because these two algorithms are more orientated around sharing CPU resources according to time or priority rather than attempting to gain maximum throughput like STR, making it a more appropriate choice when maximum throughput is not essential.

The data shown in 1 shows that the difference in mean duration increases between the shorter tests (1 & 2) and the longer test 3. Looking at the results shown in figure 5 we can see that Round Robin had the highest first job finish time. These two points show that the mean duration is affected in Round Robin scheduling because the mean is altered due to jobs generally taking a longer time to finish than with the other scheduling algorithms. It is suggested that because jobs a cycled through in the order they are added, the shorter jobs are not finished as quickly as with other approaches as they are constantly waiting for time on the CPU and hence upping their duration and therefore the mean duration.

Finally, Lottery scheduling shares similar issues with Round Robin in the test of mean duration. Lottery scheduling has a higher mean duration that STR and FCFS because the its approach to scheduling uses randomness and the priority of the job (which has nothing to do with job size) to select which job should get processing time next. This means that, like Round Robin, shorter jobs may not be prioritised CPU time and therefore will generally take longer than with STR. However, unlike Round Robin, CPU is not shared equally among all jobs. This coupled with the element of randomness ensuring that all jobs are potentially allotted processing time makes the likely mean duration of Lottery scheduling difficult to quantify, but generally appears to be slightly quicker than with Round Robin in the average case, but because of Lottery scheduling being non-deterministic the performance may vary per run.

4.3 Overall Processing Time
When comparing the overall processing time taken to complete the tests as seen in figure 2, Round Robin appears to be marginally better than the other algorithms where it either equals or surpasses every other scheduler in CPU time. If we compare with evidence with the data shown in figure 4 it can bee seen that Round Robin wastes a very small amount of time being idle compared with the other three algorithms. This suggests that while jobs scheduled using Round Robin take (on average) longer to complete Round Robin is has much more efficient time use. This coincides

with expectations because Round Robin does not waste time with jobs that have been blocked for I/O but will continue cycling through other available jobs.

Lottery scheduling was the second best performer for overall with most values CPU usage times falling between those of Round Robin and FCFS. The explanation for this can again be seen by turning to 4 where it can be seen that lottery scheduling had the second lowest idle time of any scheduler, suggesting that there is positive correlation between the amount of total overall processing time for a scheduler and the amount of time it spends idle.

In contrast, the worst performing scheduler in this category was Shortest Time Remaining which had the highest total CPU time in every test except test 3. Comparing this with the results of the idle testing we can see that the correlation between processing time and idle time holds once again. In particular, on the high number of blocking jobs test STR spent a massive 31.7% of its total execution time being idle; causing a massive waste of system resources.

According to the data in figure 2 all of the scheduling algorithms performed exactly the same on the test with no I/O blocking which strongly agrees with the prior argument that the major factor affecting the overall processing for a set of jobs is what a scheduling algorithm does when a job is blocked for I/O.

Interestingly, there also appears to be a negative correlation between the number of context switches a scheduler makes and its overall processing time. Figure 3 shows that Round Robin had the most context switches in every category while STR consistently had the lowest number of context switches. Both Lottery scheduling and FCFS also appear to fit with this trend. However, it appears that the number of context switches made is mostly lower for all scheduling algorithms on the test with a high amount of I/O blocking. This could be explained by the fact that the high I/O blocking test blocks jobs for a long period of time rather than making many short blocks meaning that the schedulers are able to make a single context switch and forget about the job until it returns. By contrast test 3 showed a high number of context switches due to having many jobs that required being block for a small portion of time.

Therefore, in summary of this section, it appears for the collected data that there is a positive correlation between the overall processing time of a set of jobs and the the idle time as well as a negative correlation between the overall processing time and the number of context switches a scheduler makes. It is suggested that making more context switches reduces the amount of time that a scheduler spends being idle and hence the faster it finishes processing. However it is worth noting that the data collected from the simulated runs in no way accounts for any overheads that might occur when performing a context switch which could be considerable depending on the system and implementation.

5. CONCLUSIONS
To summarise what has been illustrated by the data analysed in the preceding sections of this document, it can be seen that there are many key factors affecting scheduling

algorithm performance. The way that a scheduler manages these factors defines the type of scheduling problem it can be used to solve but the correct solution to any problem will always remain implementation specific.

Shortest Time Remaining appears to be best suited to applications that require short processes to be handled as quickly as possible to allow for a high level of throughput and a minimisation on the average job waiting time. STR generally has a very low number of context switches compared with the other algorithms listed here meaning that in real world applications the overheads associated with a context switch would be kept to a premium. However, these benefits come at the cost of a higher amount of idle time and therefore a higher overall processing time is more likely. The fact that shorter jobs will also be brought closer to the front of the queue means that under a heavy load longer process are likely to become starved of processing time.

First Come, First served also appears to be suited to situations that require a very low number of context switches and works well when jobs are mostly short. FCFS runs into issues when longer jobs which do not get blocked for I/O are present in the queue as they they begin to hold the CPU's time. However it does have the advantage of STR accounting for starvation providing all jobs will eventually complete (although the waiting time may be very high) as every job will get to execute. There may however, be the illusion of starvation if a very long job (such as a process that is constantly left running for days) is added to a FCFS scheduler.

Round Robin scheduling has a slower mean job duration as jobs spend longer waiting in the queue but in these tests proved to be slightly faster than the other algorithms due the the low amount of time spent idle. Unlike STR, Round Robin solves the problem of starvation because every process is given an equal amount of execution time meaning that both long and short jobs receive processing time, even in a system under heavy load. However, the largest problem with Round Robin is the high number of context switching overhead as jobs get switched after every CPU cycle.

An interesting area for further investigation with Round Robin scheduling would be to up the amount of CPU cycles a job gets before it is swapped to the back of the queue. This would produce a drop in the number of context switches the scheduler has to make thereby reducing the overhead. However, but allowing multiple CPU cycles it could cause scheduler to begin to perform more like FCFS because (if the time slice is high enough) the whole job could finish before being swapped out meaning that the first job in the queue will generally be the first to finish. Therefore the key to the optimum performance of a Round Robin algorithm is likely to be finding the "sweet spot" that minimising the number of context switches while still providing jobs with fair access to processing time.

Lottery scheduling is the only scheduling approach of the three that uses the priority supplied with each test file to select which job should be processed next. The implementation used here is weighted meaning that more important jobs are more likely to be selected. This implementation is also pre-emptive, meaning that a lottery is run every time a job is added and also when a job is returned meaning that it behaves (as shown in the test data) generally like Round Robin. The weighting means that less context switches are made and therefore jobs appear to get a higher slice of CPU time in proportion to there weighting.

However, lottery scheduling can be made to work like other scheduling algorithms depending on how it is tuned. For example, it could easily be make to work in a similar fashion to Shortest Time Remaining if the weights of the jobs were based on the amount of time left before the job completes but would not be exactly the same because there is still a chance that job with a longer time could jump to the front. Therefore while the exact throughput speed offered with STR could never be achieved, Lottery scheduling can be made to approximate closely to STR but with the added benefit of solving the issue of starvation as there is always a chance jobs with higher time remaining to taking over. This will of course mean that it would also still have a higher context switching overhead than STR.

The performance of Lottery scheduling can vary between runs. This means that while it can be made to approximate other algorithms the outcome of scheduling can be highly variable. In the data presented in section 3 the average result was presented, but the actual value can vary depending on the the random value generated. Therefore while the algorithm will generally run within a couple of cycles of the mean value it can be much worse or much better. This means that Lottery scheduling may not be the best approach to take when consistent performance is key to scheduling success. It is also worth noting that the implementation used here relies on the Java's own pseudo-random number generator and so may not produce truly random results over time. However for simple purposes a pseudo-random generator should give sufficiently random results.

6.   EXECUTIVE SUMMARY
In summary of this document, it can be seen that there are a variety of different approaches that can be taken to the problem of scheduling. Each of the algorithms in this document aim to handle factors affecting the performance of scheduling in different ways.

Round Robin cycles through the jobs to give each job a slice of time on the CPU. In doing this Round Robin minimise the amount of time that it spends being idle, but at the cost of a slightly higher mean job duration and a higher number of context switches.

Similarly the Lottery scheduling implementation performed in a way that was similar to Round Robin, but because of weights allocated to higher priority jobs had a lower number of context switches and was slightly faster than Round Robin.

In contrast Shortest Time Remaining showed a high throughput and hence a slightly lower mean duration but its overall processing time proved to not be significantly better than the other schedulers due to long periods spent being idle. STR also exhibited a much lower number of context switches and hence would have a lower computation overhead in a real world application.

First Come, First Served generally performed like Shortest Time Remaining but generally had a slower mean duration as longer processes tended to hold the CPU because of the lack of job prioritisation. The lack of prioritisation caused it to have idle times that were similar in scale to STR as well as a smaller number of context switches.

In conclusion this document has shown that there is a trade off between the average job duration and the number of context switches made against the amount of time spent being idle and the possibility of process starvation. The choice of scheduler relies heavily on the amount to which these factors affect the larger system as a whole.

7. ACKNOWLEDGEMENTS

8. REFERENCES

[1] J. W. M. David Petrou and G. A. Gibson. Implementing lottery scheduling: Matching the specializations in traditional schedulers. *USENIX*, June 1999.

[2] J. L. Donaldson. Cpu scheduling. `http://www.cs. oberlin.edu/~jdonalds/341/lecture08.html`, 2010.