Report 3: Lottery Scheduling: Flexible Proportional-Share Resource Management

Dalia Camacho García-Formentí,
Instituto Tecnolgico Autnomo de México (ITAM)
Mexico City, Mexico

I. Summary

In the article 'Lottery Scheduling: Flexible Proportional-Share Resource Management' [1], Waldspurger and Weihl present a randomized resource allocation mechanism called *lottery scheduling*. The lottery scheduling approach was developed to address the lack of scheduling control in multithreaded systems without using priority schemes, which are often not generalisable, difficult to understand, and generate a large computational overhead.

The general idea of lottery scheduling is that every client has a share of lottery tickets, and every 10 milliseconds a lottery is held and the winning ticket is randomly chosen over all the tickets available. Computational resources are then given to the owner of the winning ticket. The aim of this approach is to have a fair allocation of resources that is easy to implement and avoids starvation.

The lottery tickets can have additional properties if treated as objects that can be transferred in messages. These additional properties may help improve the performance or the fairness in mutually trusting environments. These properties are *ticket transfers*, *ticket inflation*, *ticket currencies*, and *compensation tickets*.

A ticket transfer occurs when a client sends its tickets to another client, this may occur when the client sending the tickets is blocked and if it was given computational resources it would not be able to continue with its execution. By doing ticket transfers only processes that are able to execute are considered in the lottery.

Ticket inflation occurs when a client creates more lottery tickets, thus having a higher probability of receiving computational resources in the next lottery. If clients are selfish, this will not help fairness. However, in mutually trusting environments this can improve performance. Moreover, some clients may destroy their tickets if they are blocked, which eliminates the need of ticket transfers.

Ticket currencies can be established within trust groups, this helps diminish the effect of inflation and deflation outside of trust groups.

Finally compensation tickets are given to clients that use only a fraction of the resources they had. This ensures that light processes use the same amount of resources as heavy processes, which grants fairness.

The authors implemented lottery scheduling on a Mach 3.0 microkernel with a scheduling quantum of 100 milliseconds. In terms of randomization they were able to implement the Park-Miller algorithm with only 10 RISC instructions. To find the winning ticket a search algorithm was needed, if clients are on a list this procedure is $O(n)$, but if tree structures are used this procedure is only $O(\log(n))$. They exported an interface to the microkernel, that contained all the necessary operations for ticket management that support the properties previously stated. They also created a user interface that enabled the user to create, send, and destroy tickets and currencies.

Several experiments were held to evaluate fairness in different scenarios. The first experiment consisted on trying different ratios of ticket shares between two processes, the results show that the time given to each process is consistent with the ratio of tickets. The second experiment consisted on having different Monte Carlo simulations simultaneously, whose ticket distribution was dependent on the error of each simulation. The reason for this was that if a new experiment was launched it could easily keep up with older experiments. The third experiment had to do with queries done to a server by three threads with different priority. The thread with the highest priority finished first as expected. The fourth experiment evaluated if changes in priority affect resource allocation accordingly. The fifth experiment showed how ticket inflation within a task does not affect the amount of resources given to a second task. Finally they evaluated the overhead produced by the lottery scheduling. They found the lottery scheduling required 2.7% less operations and was 1.7% faster than the unchanged Mach version.

The lottery scheduling was also evaluated for concurrent threads that require synchronisation. If a thread acquires a lock of a variable v, threads that may need to access v will transfer its tickets in order that the thread with the lock finishes first so they can continue.

Some adaptations to the lottery scheduling can be implemented, such as the inverse lottery in which a loser is randomly chosen and it must give up its resources. Also, additional management techniques can be used when there are several computational resources to compete for.

II. Analysis

Waldspurger and Weihl have experimentally shown that in general lottery scheduling achieves fairness in different applications, as long as the proportion of tickets each client holds is consistent with the priority of each process.

Still, there are some issues that should be considered. In terms of the fairness they considered ratios between two processes up to 10:1, where the observed ratio was in fact 13.42:1. They argue that in the long term the resource allocation tends to the ticket ratio. However, if larger differences in ratios are observed the time that it takes to achieve fairness may be much longer, and the absence of starvation is not granted. For example if the ratio is 1,000:1 and the quantum scheduling is every 100 milliseconds, in average it would take 100 seconds for the low priority process to receive resources. Thus, it is also important to consider thresholds to the priority ratios.

Along these lines, it is also important to remember that a mutually trusting environment is assumed. If this is not the case clients may not be willing to cooperate with each other. In a non-cooperative scenario ticket transfers and ticket deflation would not occur, and each client would try to inflate its tickets as much as possible. Then, processes that are on hold may still receive resources and fairness may not hold.

Another case that could be problematic is that in which a client inflates its tickets by acquiring multiple compensation tickets. Assume there are two clients with an even amount of tickets, one of them is selected and uses only a proportion of the resources, then it receives more tickets, which increases its probability to be selected. If this same client is continuously selected, then the share of tickets it holds constantly increases, reducing the probability that the other client is selected.

One more thing to consider is that the lottery scheduler was not the only scheduler working throughout their experiments. They claimed that for high priority threads they used the previous priority scheduler. This does not mean lottery scheduler cannot be the only scheduler used, but it implies that establishing the share of lottery tickets for each thread is not easy to define.

In conclusion lottery scheduling is easy to understand and implement, but assigning the correct amount of tickets to each thread is not a trivial task. Furthermor, one must ensure lottery scheduling is implemented in a mutually trusting environment.

References

[1] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, (Berkeley, CA, USA), USENIX Association, 1994.