Paper title: The Battle of the Schedulers: FreeBSD ULE vs Linux CFS
Paper authors: J. Bouron, S. Chevalley, B. Lepers, W. Zwaenepoel, R. Gouicem, J. Lawall, G. Muller, J. Sopena
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
This paper considers the two open-source schedulers - FreeBSD's ULE and Linux's CFS scheduler. Each of these differs in terms of design and implementation and this difference affects application performance under different workloads. It is difficult to compare the application performance for the two as it is closely intertwined with the OS subsystems, which are different for FreeBSD and Linux.

There are several similarities as well as differences between the two schedulers. While ULE is simple while CFS is more complex. Load balancing is different too - ULE regularizes the number of threads per core while CFS balances the load across cores. FreeBSD uses FIFO runqueues while Linux uses priority for threads on its runqueues. These differences cause the schedulers to affect the performance in both per-core scheduling as well as load balancing scenarios.

Previously, **Abaffy** compared the waiting time of threads in scheduler runqueues, while **Schneider** compared the performance of the network stack of the two OS. However, instead of comparing the two OS, this paper attempts to independently study the two schedulers by porting ULE to Linux. There are several other works that highlight different aspects of these schedulers, but are not as comprehensive as this one.

What are the key contributions of the paper?
Given the differences between the two schedulers, there is a variance in the performance of various applications under varying workloads. This paper attempts to analyze these differences and outlines the specific scenarios where each scheduler, owing to their design, has an upper hand.

Firstly, to perform a fair comparison, the authors ported the ULE scheduler to Linux and used it as a default scheduler to run all threads on the machine. The functions of the ULE scheduler were changed to their corresponding commands in Linux. The authors demonstrate that ULE gives interactive threads a higher priority and this might lead to starvation of the non-interactive threads, even in single application workloads. However, this starvation has its own benefits in certain cases. In terms of load balancing, CFS is faster but doesn't achieve perfect balance while ULE attains perfect balance but is slower.

These observations made by the authors help in outlining particular scenarios where each scheduler can perform better.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.

When evaluating the per-core scheduling performance, it was observed that CFS prefers fairness for all threads, while the ULE prioritizes interactive threads over batch threads. The paper considers a multi-application workload where one of the applications is compute-intensive (fibo) while the other one is mostly asleep (sysbench). Under CFS, fairness is chosen and so both applications share the resources. This results in a longer completion time for sysbench as both applications share the resources. Under ULE, however, sysbench is classified as an interactive application resulting in no shared resources and early completion, after which the compute-intensive thread is allowed to run alone. Since applications can run alone, they are able to use the cache more efficiently and are faster on ULE as compared to CFS. This is particularly beneficial for latency-sensitive applications. Further, for single-application workloads, the starvation caused by ULE helps in avoiding oversubscription, and allows applications to perform better when all the threads are fighting for the same task. Applications whose threads do not sleep do not have any impact of starvation.

Next is the analysis of load balancing. In terms of the time taken to balance a static workload on all cores, ULE takes longer as the load balancer only migrates one thread at a time. In contrast, load balancing happens much faster on CFS although it never achieves perfect load balance as it only attempts to balance the load when the imbalance is significant. In terms of thread placement, ULE takes a long time because of a cascading barrier between the threads which prevents batch threads from waking up other threads. CFS is fair and quickly wakes up the threads, but fails to achieve perfect load balance. Altogether, ULE performs 2.75% better than CFS.

Write down one question you may want to ask the authors.
If CFS was programmed to attain perfect balance, what would be the extent in which it would positively affect multi-application workloads?