# The Linux Scheduler: Complicatedly Simple Calculations

Kurt Slagle, Dustin Hauptman
{kslagle,dhauptman}@ku.edu
University of Kansas, USA

Abstract—The Linux scheduler is written to be adaptive, flexible, and customizable. However, with this modularity comes complexity. As the kernel cannot easily perform floating point computations, there are present, inside the scheduler's per-task computations, a number of operations that are seemingly overly complex in an attempt to reproduce the floating point operations using fixed-point operations. This paper looks in-depth at one function called repeatedly inside the kernels Completely Fair Scheduler, and presents a statistical analysis of the values passed to and from this function, as well as the accuracy of the performed computations.

## I. Introduction

The concept of the Completely Fair Scheduler (CFS) is deceptively simple. The source code for this inside the Linux kernel appears, initially, far more complex than would seem necessary. For example in Figure 1, the code shown simply calculates the new delta_exec by multiplying it with the weight and inverted load weight. The purpose of implementing the code in this way is not immediately clear and not elaborated on in detail.

We seek to analyze the benefits gained or loss by implementing the code in this way.

## II. Background

Cite a paper [2].

Cite multiple papers [1], [2]

## III. Your System

## IV. Evaluation

...

## V. Conclusion

...

## References

[1] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In Operating Systems Design and Implementation (OSDI), volume 99, pages 45–58. USENIX, 1999.

[2] A. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synthesis Lectures on Computer Architecture, 4(1):1–108, 2009.

```
static u64 __calc_delta(u64 delta_exec,
unsigned long weight, struct load_weight *lw)
{
        u64 fact = scale_load_down(weight);
        int shift = WMULT_SHIFT;

        __update_inv_weight(lw);

        if (unlikely(fact >> 32)) {
                while (fact >> 32) {
                        fact >>= 1;
                        shift--;
                }
        }

        /* hint to use a 32x32->64 mul */
        fact = (u64)(u32)fact * lw->inv_weight;

        while (fact >> 32) {
                fact >>= 1;
                shift--;
        }

        return mul_u64_u32_shr(delta_exec,
                                fact, shift);
}
```

Fig. 1: ___calc_delta inside fair.c