Assessing the Sustainability of Local Businesses: A Data-Driven Approach Incorporating Geographic Factors

Abstract

This project aims to develop a comprehensive framework for evaluating the sustainability of businesses within a specified geographic area. The aim is to identify and prioritize businesses that hold significant community value, taking into consideration factors such as popularity, distinctiveness, age/history, and geographic context. A robust dataset comprising businesses from California will be leveraged for this analysis, obtained from sources including Yelp, Google Local Reviews and BuzzFile.

ACM Reference Format:
. 2024. Assessing the Sustainability of Local Businesses: A Data-Driven Approach Incorporating Geographic Factors. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Background

In today's dynamic economic landscape, the sustainability of local businesses is important to the health and growth of communities. Business sustainability refers to the capacity of an enterprise to endure and thrive over the long term, while simultaneously contributing positively to the social, economic, and environmental well-being of its surroundings. Several factors influence the sustainability of a business, each bearing unique significance to its operations. These factors can be broadly categorized into internal and external elements. For the scope of this project, we will only be focusing on the external factors.

1.1 Community Engagement

An integral aspect of a sustainable business lies in its ability to forge meaningful connections with the community it serves. Active engagement, inclusive practices, and contributions to the cultural fabric of the locality are central to a business's role as a valued community asset. One of the major sources of community engagement is present in the form of reviews and rating. Customers often mention interactions with the community in their reviews. They may highlight events hosted by the business, partnerships with local organizations, or the business's role in community initiatives. These mentions provide tangible evidence of the business's engagement with the local community.

We intend to incorporate review data by extracting both the emotional significance and valence conveyed within each review. Additionally, we will cross-reference the review sentiments with the population data of the corresponding zip code. This combined approach will enable us to generate two distinct scores: one reflecting the emotional impact and the other representing the overall assessment of the business.

For the emotional significance, we found a paper "A review on sentiment analysis and emotion detection from text", where the author use two different approaches, Lexicon based approach and Machine Learning based approach. The Lexicon based approach is a keyword-based search approach that searches for emotion keywords assigned to some psychological states. They suggested an effective strategy to obtain word-level emotion distribution to assign emotions with intensities to the sentiment words by merging a dimensional dictionary named NRC-Valence arousal dominance. This particular method capture the intensity (Love and joy, each signifies a positive valence but love has more intensity compared to joy).

The author also explore on machine learning based approach

1.2 Linux CFS Scheduler

Linux CFS (Completely Fair Scheduler) is an advanced process scheduler that prioritizes fairness and efficient use of resources. It uses the concept of `vruntime` and `min_vruntime` to distribute CPU time fairly among processes. CFS' load balancing mechanism is also very fast, that can quickly adjust the distribution of processes across CPU cores to ensure balanced utilization and optimal performance.

One of the main advantages of CFS is its ability to make the process fair. It is intended to allocate CPU time proportionally based on priority and resource requirements, and to prevent a single process from monopolizing system resources. However, fine-tuning CFS for a particular workload can be complex. Proper configuration and parameter tuning require a deep understanding of the algorithm and system characteristics. Careful tuning is required to ensure optimal performance and avoid suboptimal configurations that can affect system efficiency.

1.3 FreeBSD ULE Scheduler

The ULE scheduler is the default process scheduler in FreeBSD. It features a multi-level response queuing system for efficient process prioritization. It categorizes the processes into interactive and batch, based on an interactivity score. These separate categories are maintained in separate runqueues. The interactive tasks are given absolute priority over batch tasks. So, it provides maximum responsiveness when an interactive task enters its runqueue.

The absolute priority of interactive tasks over batch tasks can be disadvantageous. If an interactive task can keep a CPU core busy, ULE will completely starve a batch task that is waiting to be executed. Another disadvantage of ULE is that it takes a lot of time to distribute the load across the CPU cores when there is a burst of activity. This overhead can affect system performance and responsiveness, especially in situations with frequent load balancing or large numbers of CPU cores.

2 Implementation

In this section, we describe how we designed SAM by modifying the CFS scheduler to use ULE's interactivity scoring mechanism.

In CFS, the vruntime is the total execution time of a thread adjusted according to its priority by the scheduler as shown in Eq. 1. For example, a task with 0 nice value will have its vruntime equal to its total execution time. Whereas, vruntime will be lower than the total execution time for higher priority tasks and greater for lower priority tasks. This is the value CFS uses to pick the next task to run.

$$\text{vruntime} \mathrel{+}= \text{delta\_exec} * \frac{\text{nice0\_weight}}{\text{curr\_weight}} \tag{1}$$

where,

delta_exec	*time elapsed since last update,*
nice0_weight	*weight of default priority tasks,*
curr_weight	*weight of the current task*

In SAM we attempt to use the interactivity score instead to pick the next task to run. We calculate the interactivity score for each task using Eq. 2. This results in scores in the lower half of the interactive score range for threads that spend more time sleeping than running (typically interactive tasks), and scores in the latter half for threads that spend more time running than voluntarily sleeping (typically batch tasks). By using the vruntime adjusted by CFS to calculate the interactivity score instead of the total runtime, we are making sure that the task's priority will be taken into consideration.

$$\text{scaling\_factor (m)} = \frac{\text{max\_interactivity\_score}}{2} \tag{2}$$

$$\text{interactivity\_score (I)} = \begin{cases} \dfrac{m}{(s\,/\,r)} & \text{if } s > r \\ m + \dfrac{m}{(r\,/\,s)} & \text{otherwise} \end{cases} \tag{3}$$

where,

s	*voluntary sleeping time of the thread,*
r	*time spent running*

We modify the CFS code to use the interactivity scores to store the tasks in the RB tree runqueue. When we pick a new task to execute, we choose the left-most task with the least interactivity score. When a new task enters the queue, we once again check the interactivity score to decide if it should preempt the current task. It is to be noted that this is a primitive implementation and can be further optimized for performance.

3 Evaluation

In this section, we perform a performance evaluation of the SAM scheduler, and describe the results. We describe the experimental setup in Section §3.1 and our experimental methodology in Sections §3.2 and §3.3 respectively.

3.1 Experimental Setup

We evaluate SAM on a 16-core AMD® Ryzen 9 5980HX machine with 16GB RAM and 2TB NVMe SSD running a Linux v5.14.0 kernel with hyper-threading enabled. We run a series of workloads and synthetic benchmarks that mimic a real-word commodity PC/laptop.

3.2 Single-core scheduling

Since SAM was inspired by the work done by *Bouron et al.*[3], we begin our evaluation by comparing the performance of CFS and SAM running a multi-application workload consisting of one compute-intensive task (fibo, which recursively computes fibonacci numbers), and one interactive application (sysbench, filesystem I/O, with 350 threads). We set CPU affinity for both applications to restrict their execution to a single CPU. We used 350 threads in the interactive application to completely saturate the core and mimic contention. On a machine without contention, both fibo and sysbench individually take 85s to execute to completion. Figure 1 shows the evolution of cumulative run-time for each application under CFS (a) and SAM (b).

In CFS, the CPU is approximately evenly shared between the two applications. Once sysbench completes execution (108s), fibo progresses at double the pace and finishes at 141s. CFS tries to achieve a fair schedule.

In SAM, the interactive task is clearly favored without foregoing the fairness of the batch task. In comparison with CFS,

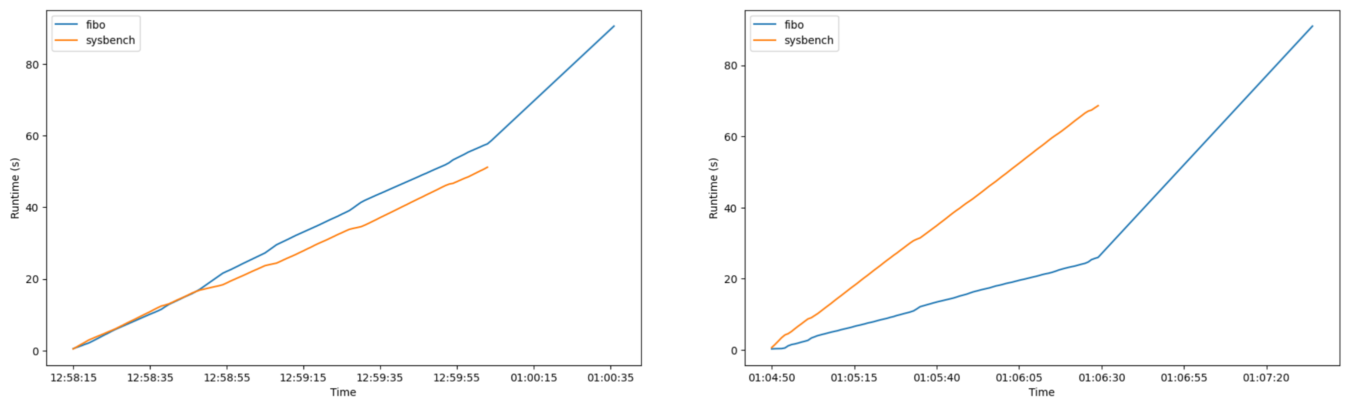

Figure 1. Accumulated CPU runtime of a batch task (fibo) and an interactive task (sysbench), both pinned to one core, saturating the entire core. CFS tries to be fair and allocates 50% of CPU to each task. SAM favors sysbench more, but doesn't fully starve fibo.

sysbench completes earlier at 96s, after which fibo progresses rapidly without contention and finishes at 164s. This shows an improvement in interactivity responsiveness by 11%. Juxtaposing this with ULE's performance in *Bouron et al.* [3], it is worth noting that in ULE the batch task is completely starved until the interactive task is completed, whereas in SAM, the batch task progresses albeit at a slower pace. This observation is to be taken with a grain of salt given that the workloads and evaluation environments are different.

3.3 Real-world benchmarks

To assess the performance of SAM on real-world tasks, we run UnixBench [2] with parallelism 1 and 16, on all of the 16 cores. We run 6 applications from the Phoronix test suite [5], namely, compilation benchmarks (build-apache), compression (7zip), image processing (c-ray), scientific (himeno), cryptography (john-the-ripper) and web (apache). These tests are similar to the ones that were performed in [3]. We also repeated the same tests with unmodified CFS to get the baseline performance.

Figure 2 shows the relative performance of SAM with respect to CFS's baseline performance for UnixBench and Figure 3 shows the relative performance for the Phoronix tests. It can be observed that it does better in some tasks and worse in some, but overall SAM achieves comparable performance to CFS. We also observed that SAM performed slightly better with multiple cores than a single core.

4 Related Work

There have been several attempts in the past to port features of ULE to CFS or vice versa [1]. We want to highlight some of those attempts that we took inspiration from in this section.

In 2009, a CFS patch with some of the ULE features such as cache affinity and run queues sorted by priority named "Brain Fuck Scheduler" was released by developer Con Kolivas [6]. It was one of the earliest demonstrations that ULE features can be added to the CFS scheduler.

In 2020, a patch implementing a ULE-inspired load balancing mechanism was proposed to the CFS scheduler. It used an RB tree to track the task's cache affinity to improve load-balancing performance, similar to ULE. This patch was accepted and merged into the Linux kernel in version 5.8.

Finally, we look at CacULE [4], a CFS patchset by developer Hamad Al Marri inspired by ULE's interactivity scoring mechanism. We share the same goal of enhancing system responsiveness in CFS. They achieve this by calculating the interactivity scores for each task like in ULE to be used in selecting the next task to run. They also replaced the RB tree structure of the task runqueue with a linkedlist which incurs an O(n) runtime complexity every time the scheduler has to pick a task to run. We take inspiration from their porting of the ULE scoring mechanism to CFS but stick to the existing RBtree structures used by CFS.

5 Conclusion

Achieving high responsiveness to interactivity while ensuring fairness to all the tasks is very hard. In this paper, we present a new scheduler SAM that addresses the limitations of two of the most widely-used open-source Operating Systems' default schedulers - CFS and ULE. While CFS is fair to all processes, it's responsiveness is not as quick as ULE. On the other hand, ULE achieves high responsiveness but might do that by starving batch tasks. SAM combines the best of both worlds by introducing the interactivity scoring mechanism of ULE to CFS. Our experimental results show

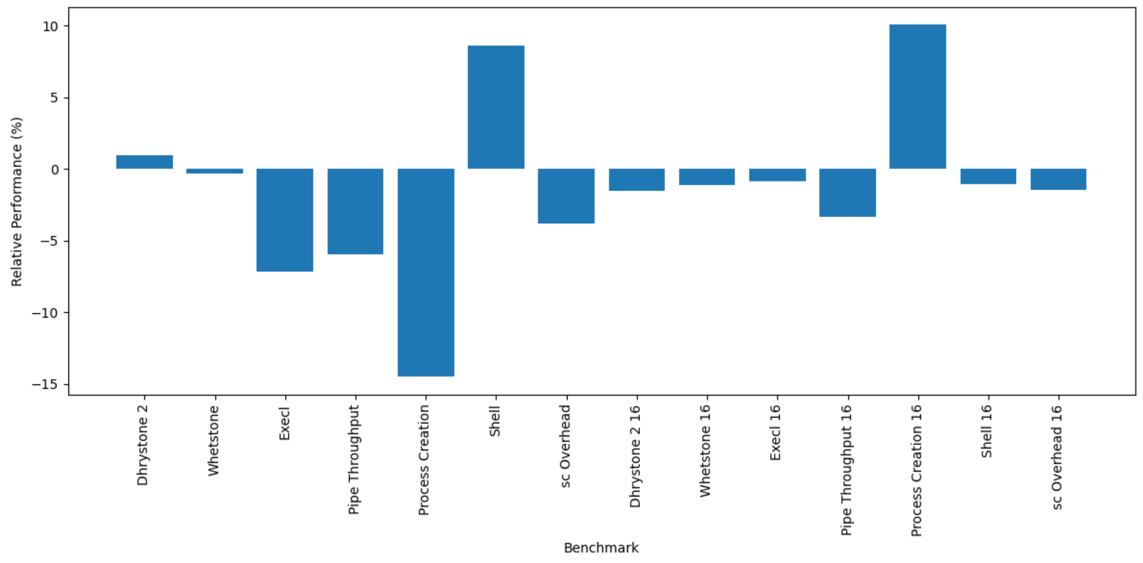

Figure 2. Relative performance of SAM on UnixBench [2] compared to CFS. The y-axis shows relative performance gain/drop

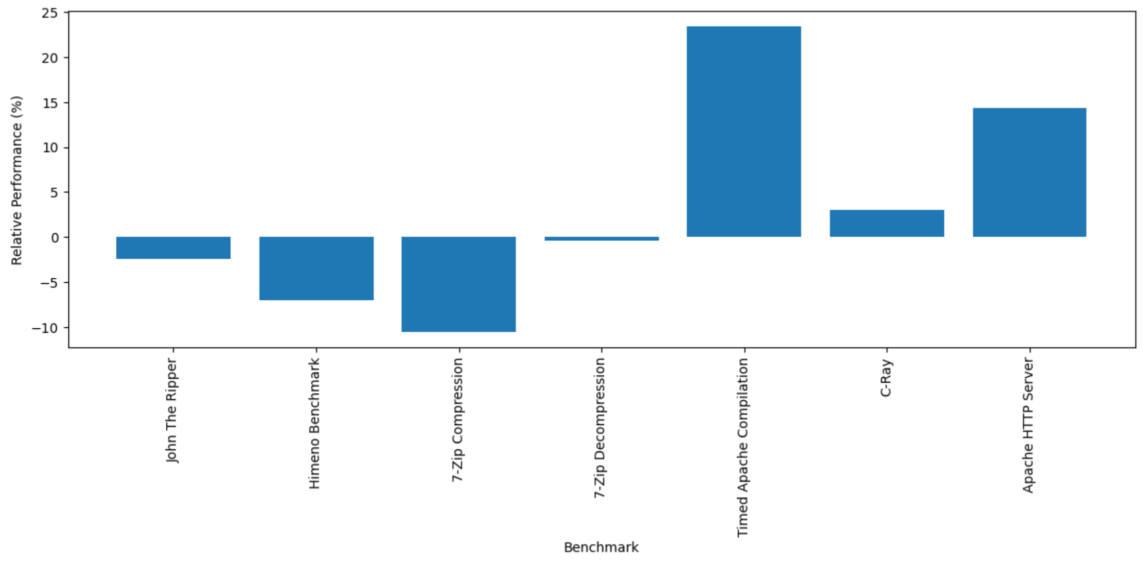

Figure 3. Relative performance of SAM on Phoronix [5] compared to CFS. The y-axis shows relative performance gain/drop

that SAM is able to achieve 11% improved responsiveness to interactivity while maintaining comparable performance to CFS for real-world applications. We see a lot of scope for improvement in SAM's performance and hope to work on it in the future. This was our humble attempt to create a fair scheduler with enhanced interactivity.

Acknowledgments

Prof. Dongyoon Lee, for the wonderful course content and inspiring us to take on this project.

References

[1] Jaroslav ABAFFY and Tibor KRAJČOVIČ. 2009. Latencies in Linux and FreeBSD kernels with different schedulers – O(1), CFS, 4BSD, ULE. In *Proceedings of the 2nd International Multi-Conference on Engineering and Technological Innovation*. 110–115. https://www.iiis.org/cds2008/cd2009sci/CCCT2009/PapersPdf/T794MV.pdf

[2] Tom Yager Ben Smith, Rick Grehan. 2023. *UnixBench*. https://github.com/kdlucas/byte-unixbench

[3] Justinien Bouron, Sebastien Chevalley, Baptiste Lepers, Willy Zwaenepoel, Redha Gouicem, Julia Lawall, Gilles Muller, and Julien Sopena. 2018. The Battle of the Schedulers: FreeBSD ULE vs. Linux CFS. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 85–96. https://www.usenix.org/conference/atc18/presentation/bouron

[4] Hamad Al Marri. 2021. *cacule.* https://github.com/hamadmarri/cacule-cpu-scheduler

[5] Phoronix Media. 2023. *Phoronix Test Suite.* https://www.phoronix.com

[6] Wikipedia contributors. 2023. Brain Fuck Scheduler — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Brain_Fuck_Scheduler&oldid=1140833784 [Online; accessed 8-May-2023].