# Scheduler-Aware Virtual Memory Management

Emery Berger, Scott Kaplan†, Bhuvan Urgaonkar,
Pritesh Sharma, Abhishek Chandra, Prashant Shenoy
University of Massachusetts †Amherst College
Amherst, MA Amherst, MA

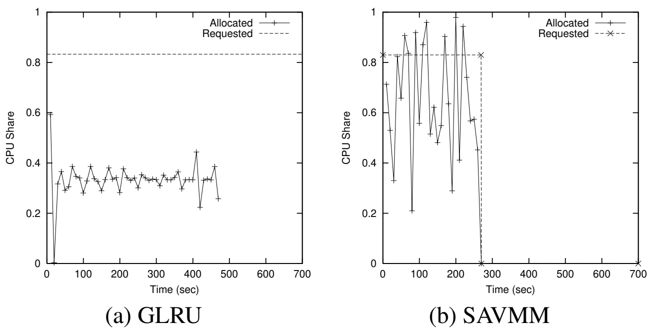
## 1. INTRODUCTION

Virtual memory management has come to play an increasingly adversarial role in the scheduling of processes. While it is the job of the CPU scheduler to determine the order and length of execution of processes, its attempts to meet the desired scheduling constraints are thwarted by the memory manager under memory pressure. This is because scheduler policies like priority are not supported by memory managers, so that heavy paging under memory pressure can result in non-deterministic divergence from the desired CPU allocations. For instance, the memory access pattern of a low-priority process could cause a higher-priority process to lose its pages and page fault frequently, thus inducing a form of priority inversion.

This effect is the result of two problems with traditional virtual memory managers. First, they are *scheduler-oblivious*, i.e., they are neither aware of scheduler policies, nor do they attempt to cooperate with the scheduler. Second, their primary objective is to maximize system-wide throughput by maximizing the overall page faults. This approach prevents the meeting of scheduler goals in the presence of memory pressure.

In this work, we describe SAVMM – the first *scheduler-aware virtual memory manager* that we are aware of. SAVMM works by formalizing the notion of scheduler-awareness as a utility maximization problem, where the utility of a process is based on its memory requirement as well as its scheduler-defined importance.

## 2. ALGORITHM AND RESULTS




(a) GLRU (b) SAVMM

**Figure 1: CPU shares received by a process using different memory managers: SAVMM closely adheres to the scheduler goals while GLRU ignores them.**

Our virtual memory management algorithm is especially effective at meeting the scheduling needs of a proportional-share CPU scheduler. The goal of such a scheduler is to allocate CPU time among processes based on their requested shares. Note that simply sharing the main memory proportionately (as done by Waldspurger [3]) is insufficient to achieve this goal. For instance, a process that needs to run twice as often as another may actually need ten times more memory to avoid heavy paging.

To incorporate the memory requirement of each process, we define the notion of *utility* for each process based on the number of pages it can be allocated. This utility corresponds to the normalized hit rate (fraction of in-core references) of a process for a given memory allocation. We then weigh this utility by the desired CPU share for each process.

$$U_p(n) = H_p(n) \cdot w_p,$$

where $U_p(n)$ is the utility and $H_p(n)$ is the hit rate for process $p$ with an allocation of $n$ memory pages, and a desired CPU share $w_p$. We maintain reference histograms [2] for each process that allow us to compute hit rates. Memory allocation then reduces to a utility maximization problem, where the memory allocation for each process is determined by maximizing the system-wide utility $\sum_p U_p(n)$ subject to the constraint $\sum_p n \leq M$, where $M$ is the main memory capacity in pages.

We have developed a Java-based trace-driven simulator that allows to explore the interaction between virtual memory management and CPU scheduling. We have implemented both SAVMM and Global LRU (GLRU) – the traditional memory management algorithm employed in several popular kernels such as Linux. We use SFQ [1] – a proportional-share CPU scheduling algorithm. We ran an experiment with 3 processes requesting CPU shares in the ratio 1:10:1. The memory requirement of all processes was identical. Figures 1(a) and (b) show the CPU allocation of the process with share 10 while running under GLRU and SAVMM, respectively. As can be seen from the figures, GLRU allocates about one-third of the total CPU to the process, while SAVMM on average achieves the desired proportion (83%). We are currently implementing the algorithm in the Linux kernel, and we intend to study its performance using real application workloads.

## 3. REFERENCES

[1] P. Goyal, X. Guo, and H. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. In *Proceedings of OSDI'96*, October 1996.

[2] Y. Smaragdakis, S. Kaplan, and P. R. Wilson. EELRU: Simple and Effective Adaptive Page Replacement. In *Proceedings of SIGMETRICS'99*, May 1999.

[3] C. Waldspurger. Memory Resource Management in VMWare ESX Server. In *Proceedings of OSDI'02*, Dec 2002.