MLCache: Using Adaptive Techniques to Improve the Page Cache

Jose Pazos
University of British Columbia
jpazos@cs.ubc.ca

Renato Costa
University of British Columbia
renatomc@cs.ubc.ca

ABSTRACT

Application-controlled caching is often thought to be a more efficient technique than kernel-based, global policies. However, it introduces another issue for developers: having to profile each application individually to check access patterns, while possibly getting it wrong. With the rise of machine learning, it seems natural to offload the work to learning algorithms, so that application developers are freed from the responsibility of dealing with these issues, while getting the performance benefits of having more specialized policies for their systems. In this document, we present MLCache, a proposed page cache policy that learns access patterns for different applications in order to improve cache hit ratios for subsequent runs.

1 INTRODUCTION

Application-controlled caching has been proposed as an alternative to global caching policies that might not be appropriate for different types of applications, each with different caching needs. In particular, monolithic systems like Linux lack the flexibility to meet applications' requirements. These approaches often show beneficial results, especially when incorporated with different techniques such as disk scheduling, and prefetching [1]. The main issue, however, is that application developers would suffer if they are not keen enough to be able to provide a good policy for their applications. This problem becomes especially hard when taking into account the interplay between several concurrent processes, which are constantly competing for resources. Additionally, these policies should be adaptable to different workloads, possibly changing on the fly. Thus, this design is not practical. A potential solution to this complex problem is the use of machine learning, through the use of unsupervised techniques, or supervised techniques that require minimal input from a developer. We believe that this approach should provide similar enhancements to those previously seen from application-controlled caching policies, while being substantially easier to maintain, and ideally hiding the complexities from the developer by pushing the work to the kernel.

2 PROPOSAL

We propose applying an online, adaptive, linear regression learning model to achieve application-controlled caching. We think that applying supervised learning might be too tedious for the developer, as this would defeat the purpose of detaching the developer from the policy design (it might be useful for basic schemes like consecutive writes, reads, etc.). Instead, we believe unsupervised learning, based on pattern matching or clustering should still provide performance benefits, while almost completely removing the need for data access analysis from the developer. Training should be based on real-world uses of applications, as opposed to idealized uses like sequential accesses. The required modifications can be implemented either at kernel-level or user-level. There's work to be done to figure out what overhead there is when applications give caching suggestions to the kernel, vs. the possible overhead of the kernel having to manage multiple policies for different applications. A possible compromise is setting an expected cache miss rate, and only optimizing for applications that exhibit higher miss rates than the expected. We also need to take into account the fact that this learning can be a slow process, and applying it to every single application is not necessarily an attractive approach.

3 METHODOLOGY

Our implementation will focus on the kernel-side approach of the page cache. Thus, different policies will be maintained by the kernel. The basic idea for the learning algorithm is to minimize cache misses. We plan to apply a linear bandit approach for the learning algorithm. We expect our state space to be of a plausible size for this. Online learning will be useful since we cannot predict all the types of workloads in limited amounts of time, so the algorithm should be able to adapt to new workloads. Training must be done on varied workloads which represent real use-cases of the application, such that the algorithm can learn the most common access patterns, while being able to accommodate a broad range of uses for the application.

There are a number of existing attempts at solving this problem that we can draw inspiration from [2][3], and we plan to investigate the approaches taken by different systems to find out how we can improve existing designs.

4 EVALUATION

The ideal applications that would benefit from the approach proposed in this document are long-running processes with nondeterministic access patterns. In such cases, a typical LRU caching strategy might not be optimal.

We plan to evaluate the system by comparing the performance of applications running on top of it with the performance of the same application running on an unmodified kernel. Examples of such applications are:

- A database server: depending on the access patterns, a better cache policy is likely to produce measurable performance improvements. A good example candidate for this kind of test is to run PostgreSQL with different datasets and under different kinds of load.
- A backup utility: tools like `dump(1)` and `restore(1)` typically have to scan the entire file system when generating incremental backups. They are also likely to be sensitive to modifications at the page cache layer.

- A process that does sequential reading: we plan to measure the impact of our changes on tools like `grep(1)`.

5 CONCLUSION AND FUTURE WORK

Possible enhancements to this approach would be to integrate other techniques, like disk scheduling, and prefetching. This is rather complex in practice, given the possible interplay between many different types of applications. It is not clear whether a machine learning approach would provide a performance enhancement for such a design, as the modelling complexity might incur too much overhead in the system.

REFERENCES

[1] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. 1996. Implementation and Performance of Integrated Application-controlled File Caching, Prefetching, and Disk Scheduling. *ACM Trans. Comput. Syst.* 14, 4 (Nov. 1996), 311–343. https://doi.org/10.1145/235543.235544

[2] Nimrod Megiddo and Dharmendra S. Modha. 2004. Outperforming LRU with an Adaptive Replacement Cache Algorithm. *Computer* 37, 4 (April 2004), 58–65. https://doi.org/10.1109/MC.2004.1297303

[3] Yifeng Zhu and Hong Jiang. 2007. RACE: A Robust Adaptive Caching Strategy for Buffer Cache. (2007).