Advanced Operating Systems (CS 736)

Enhanced File Prefetching

Tushar Khot **Varghese Mathew** **Priyananda Shenoy**

{tushar,vmathew,shenoy}@cs.wisc.edu

1 Problem Summary

Current file prefetching implementations are static, and are agnostic to file-type. We propose to fine tune the prefetching algorithm to make informed decisions based on the type of file being accessed. Likewise, the file-type and therefore the access pattern can be used to fine-tune the cache eviction policy to improve cache utilization.

2 Motivation

We believe that the access pattern for a certain type of file generally remains the same across differnt files of that type. This observation can be put to use in determining ideal prefetch depths for different frequently encountered file-types. We expect prefetching driven by such a file-type determined depth to outperform contemporary implementations of prefetching which are rather static. Similarly with caching, the access pattern as suggested by the file-type can hint at the likelihood of the block being accessed again. A cache replacement policy driven by this can make sure that we do not cache blocks that are unlikely to be needed.

3 Initial plan of approach

The type of file determines the access pattern (like sequential, random or piecewise sequential) and the cache replacment policy (whether the block is likely to be accessed again). And if the access is sequential, the rate at which the file is normally read can be used to compute an optimal prefetch depth for the file-type. The initial part of our project involves gathering traces to determine file access patterns and analysing them on a per-file-type basis. We then plan to use this information to propose a better policy for prefetching and cache replacement.

4 Proposed Methodology and Evaluation

- As a first step, we plan to instrument the linux kernel to yield a trace of file access patterns including prefetching and cache hit/miss patterns. Through this, we desire to attain twofold objectives
    1. Determine an offline, optimal prefetching and cache replacement policy, which will serve as a reference point for comparison in our evaluation of our own strategy.
    2. Familiarize ourselves with the prefetching code in the linux kernel.
- Next we plan to gather long sample traces over real workloads. We will use these traces to determine and generalize access patterns over file-types and compute the optimal prefetch depth and cache eviction policy statically.
- As the third step, we plan to incorporate the optimal depth so obtained into the linux kernel and evaluate resultant performance of the system, both against the offline optimal strategy and the performance obtained without our modifications.
- As a potential last step, if time permits, we will try to incorporate a feedback mechanism to adjust the offline determined ideal prefetch depth.

5 Software resources

We propose to confine ourselves to Linux 2.6.x in this study. Additionally, we will primarily work on the ext2 file system. However, if time permits, we may also perform the same modifications on the IBM jfs filesystem to exclude file-system specific anomalies.

6 Hardware resources

We plan to run our tests on an i386 system with a SATA hard disk. We plan to use a machine in the Crash and Burn Lab for our experiments.

References

[1] Behdad Esfahbod. Preload An Adaptive Prefetching Daemon. *Masters Thesis, Graduate Department of Computer Science. University of Toronto*, 2006.

[2] Jeffrey Scott Vitter, P. Krishnan. Optimal prefetching via data compression. *Journal of the ACM, vol.43, pp. 771-793*, September 1996.

[3] James Griffioen, Randy Appleton. Reducing File System Latency using a Predictive Approach *USENIX 94*, 1994.