New Cache Writeback Algorithms for Servers and Storage

Sorin Faibish Peter Bixby John Forecast Philippe Armangau Sitaram Pawar
EMC

Abstract

In this paper we propose a new paradigm and algorithms to address cache writeback performance in file systems, servers and storage arrays. As servers and storage processors move to multi-core architecture, with ever increasing memory caches, the cost of flushing these caches to disk has become a problem. The paper introduces new algorithms to change the application data writeback from using watermark or aging based flush to something that approximates the rate of the incoming application I/Os. Our proposed algorithms are applicable to local file systems and remote servers and prove that rate based cache writeback algorithms are the most efficient replacement for watermark and aging based flushing. We are in process to apply the new algorithms to Linux file systems.

1. Introduction

During the last few years we have witnessed dramatic changes in disk hardware as seen by file servers and local file systems. First, disk capacity has increased at a high pace, at times doubling each year, while the physical size of drives remained unchanged. As a result much of the classical file system metadata grouping [1], ordering [2] and caching mechanisms aimed at reducing random disk access latencies are becoming obsolete or at least have a reduced impact on the performance of file systems.

The majority of cache writeback algorithms used by storage arrays and file systems is based on some form of watermark crossing, which triggers a flush of user application dirty pages. Watermark based cache flushing is used by many database application servers to writeback the local dirty pages to disk. When a high watermark is crossed the array flushes all the dirty pages to disk at a high speed until the level goes under a low watermark when the in-coming I/Os are resumed. This large amount of I/O can flood the storage subsystem, in some cases appearing to stop the server.

In this paper we address the problem of cache writeback of user application data with a secondary goal of generalizing the new algorithms to MD as well. We investigate several algorithms that have the goal of controlling the number of application dirty pages in the buffer or page cache while preventing the incoming application I/O stream slowdown and smoother utilization of disk I/O throughput in order to maximize application performance.

2. Problem definition

As we discussed in the previous section, cache writeback or flush to disk of user application data, presents new challenges to storage systems. This is mainly due to the special characteristics of file data versus metadata. Although most file systems cache data in memory to improve performance, not all cached pages in the buffer cache of servers and arrays have the same flushing needs. Metadata cache pages can be relatively small in size and number, but are often placed randomly on disk.

Due to the larger number of user data dirty pages the I/O throughput required to flush the data is disproportionally higher than the through-put needs for flushing MD, but the MD must have higher flush priority as it may govern the flush of the user data pages. Flushing user data pages to disk is often dependent on disk updates of the corresponding cached pages of MD for the files that contain the user data pages, otherwise the file system may become inconsistent. Current MD and data flushing is triggered based on some for of high watermark crossing.

Based on observations on modern file and storage systems, it is clear that flushing using watermark techniques is incapable of ensuring even application performance as compared, to say, I/O rate based cache writeback techniques. By controlling the disk throughput the effect of large changes in the incoming I/O stream will be mitigated, thus preventing I/O bursts which will be reflected back to the application. Of course, a storm of I/Os over a very short period of time may still result in some burstiness in the write stream and slowdowns for the client.

3. Proposed solutions highlights

The algorithms proposed in this paper try to solve the flushing problem by using I/O rate proportional algorithms, with the goal of producing acceptable performance for the majority of applications. The proposed algorithms will use the first derivative of the change in number of user data dirty pages in the buffer cache with respect to time. This is because it’s difficult

to count the number of dirty cache pages generated by the application, until they actually become dirty pages in the buffer cache. This is because the size of the application I/Os may be different than the cache page size or may be unaligned with the cache page and usually full pages are cached, even if only part of their con-tent has changed. In addition, some pages may be re-written multiple times in the cache before the page is written to disk.

4. Cache writeback proposed algorithms

We developed and examined 5 different algorithms, each designed to more closely match the flush rate to the rate of incoming I/Os. In general we chose to look primarily at algorithms that are self tuning. This was because choosing tuning parameters can be difficult and may not work well in highly dynamic situations. We present discuss and simulate 5 algorithms that we applied to cache writeback of Linux OS.

4.1 Modified trickle flush

4.2 Fixed Interval Algorithm

4.3 Variable Interval Algorithm

4.4 Quantum Flush

4.5 Rate of change proportional algorithm

5. Application to Linux OS cache writeback

As we show in the paper the behavior of the dirty pages in the BC is highly non-linear and using linear algorithms can improve the performance but still not maximize the performance. In order to optimize the performance we can also use non-linear filtering algorithms inspired from digital signal processing [3].

Additional work is in progress to apply the new algorithms to improve Linux cache writeback which is currently suffering performance issues due to the problem presented here. Currently we started to look at other file systems in Linux to evaluate the algorithms used for cache writeback of application data. We wrote a small utility that samples the number of dirty pages in Linux buffer cache in order to see the behavior of the DP. We sampled the dirty pages changes during Linux kernel build on a Linux server running 2.6.18 kernel and we used different cache memory sizes: 2GB, 4GB and 8GB. Figure 1 shows the 3 cases. Then we used the data form the 8GB case as input to our simulation and applied the rate proportional algorithm. The results are presented in Figure 2.

The results in Figure 2 show that using the rate proportional algorithm we would be able to finish the build using a server with 2GB of cache in same time as on a machine with 8GB and prevent swap that resulted in build failure. Our final goal will be to prevent memory swap in the Linux by using new cache writeback algorithms.

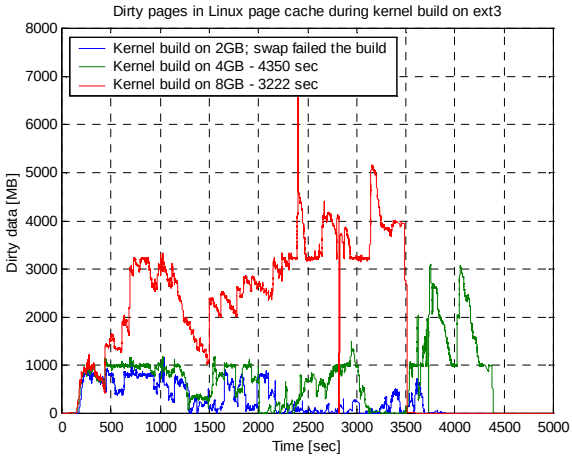

Figure 1: Kernel build on different cache sizes

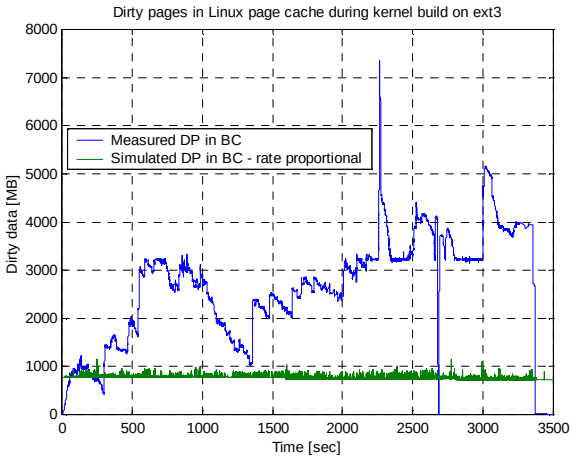

Figure 2: DP in Linux BC; measured and simulated

References

[1] Ganger, G., and Yale N. P. Metadata Update Performance in File Systems. 1994 Operating Systems Design and Implementation (OSDI), November 1994.

[2] Ganger, G. and Kaashoek, M., “Embedded Inodes and Explicit Grouping: Exploiting Disk Bandwidth for Small Files,” in Proceedings of the USENIX Winter 1997 Technical Conference, January 1997.

[3] Faibish, S., and Moscovitz, I., "A New Closed-Loop Non-Linear Filter Design", in Proceedings of the 1-st European Control Conference, Grenoble France, July, 1991.