Review #6
Locating Cache Performance Bottlenecks Using Data Profiling

A. Pesterev, N. Zeldovich, R. T. Morris

Proceedings of the EuroSys 2010 Conference, 2010

Jean-Pierre Lozi

May 6, 2010

1 Problem

Caches are the source of major performance bottlenecks, especially in the now increasingly common multi-core architectures. A range of profiling tools is available to developers, allowing them to track performance issues in their programs. However, these tools usually attribute costs to specific locations in the code, which is not always helpful when dealing with memory problems: when a poorly managed data structure is the cause of a performance issue, traditional profilers point to dozens of code parts that are considered suspicious because they are time-consuming, with only a few of them being actually related to the problem.

The authors argue that linking data cache misses to data types instead of code locations would be a much better approach that would provide more informative results, since cache misses show a greater correlation with data than with code.

2 Solution

DProf is a profiling tool that dynamically tracks indicators associated with data structures instead of code locations in order to allow for more efficient localization of cache-related bottlenecks. It offers four views of the collected data: the *Data Profile* view shows the number of cache misses for each of the most common data types; the *Miss Clarification* view shows what type of misses (due to sharing, associativity or capacity overload) are most common for each data type; the *Working Set* view indicates what data types were most active, how many of each were active at any given time and the cache associativity sets used by each data type; and the *Data Flow* view shows the most common sequences of functions that reference particular objects of a given type.

To create these views, DProf collects two kinds of data: *path traces* and *address sets*. A path trace records all accesses to a single data object. An address set is a data structures that contains the address and type of every object allocated during execution. Both of these data structures are built from the two kinds of raw data ultimately collected by DProf: *access samples* and *object access histories*. An access sample records information for randomly chosen memory-referencing instructions, and an object access history is a complete trace of all instructions that read or write a particular data object, from when it was allocated to when it was freed. To collect access samples, DProf uses AMD®'s proprietary Instruction Based Sampling (IBS), but similar facilities are available on Intel® chips.

3 Evaluation

The authors show that, in two case studies, DProf provides much more helpful information than both lock_stat (a profiling tool that reports statistics about locks) and OProfile (a traditional execution profiler).

For the first case study, they run multiple instances of memcached[1] that are accessed by remote clients.

In this experimental setup, data packets are often migrated from one core to another under high workload, which causes a huge drop in performance. This is due to a data queue in the network driver that is shared by all processes instead of being replicated on each core. DProf directly highlights a problem with the `size-1024` data structure that holds the data packets' payload. The Data Flow view also indicates which functions are called prior to this problem, making it possible to quickly find where to patch the code. Lock_stat provides much less helpful information but still allows the user to isolate the issue by indicating that the locks on the shared queue are time-consuming. It does not provide accurate information as to where the code needs fixing, however. OProfile is not very helpful either: it only ranks the offending function as the 10th most time-consuming one. Moreover running OProfile again after fixing the problem shows an improvement in most of the time-consuming functions, instead of just the one that requires fixing, which further hints that code-oriented profilers are bad at pinpointing such issues.

In the second case study, the authors run multiple Apache servers on a test machine, with remote clients repeatedly retrieving a file from the servers. The total number of requests per second drops abruptly after the request generation rate is increased beyond a certain point. The problem is due to Apache's `tcp_sock` objects getting flushed off the caches closest to the cores by incoming packets. It can be solved by limiting the size of the accept queues.

DProf clearly shows an huge increase in the working set size of `tcp_sock` objects and in the elapsed time between their allocation and their deallocation. Lock_stat does not reveal anything about the problem. Performing a differential analysis (i.e. comparing the result of two analyses, one before the problem and one afterwards) with OProfile is impossible since running it causes the test machine to go from the peak state straight into the drop off state.

Performance experiments show that DProf's overhead varies depending on several variables (IBS sampling rate, number of debug register interrupts triggered by second, etc.). However, it is usually reasonably low (<15%).

4 Benefits

DProf facilitates spotting cache management problems more efficiently than other debuggers. It can be used in combination with traditional debuggers to address complex problems. Thanks to its configurability, it is possible to lower the overhead enough to perform any analysis without interfering too much with the normal behavior of the monitored program.

5 Shortcomings

DProf has to monitor programs for a certain period of time to collect enough data: it could be hard to efficiently monitor programs whose execution time is very short. Moreover, cache miss problems are not always linked to particular data types, they could for instance be associated with a small subset of objects of a particular type: in that case, aggregating the data by type would add too much noise for the result to be informative.

[1]Memcached is a general-purpose distributed memory caching system often used to speed up dynamic database-driven websites.