Seesaw: QoS in Distributed Cache Systems

Brian Schwedock (bschwedo), Graham Gobieski (ggobiesk),
Elliot Lockerman (elockerm)
15-712: Advanced OS and Distributed Systems, Fall 2017

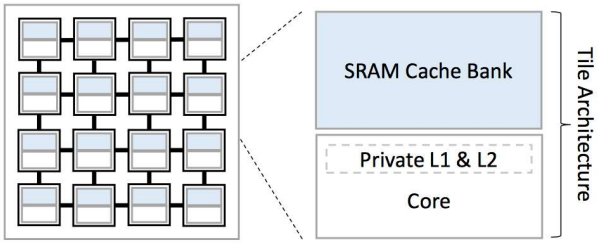
1 Abstract

High server utilization requires running heterogeneous workloads (such as batch and latency-critical applications) on the same machine, introducing the potential for interference in the shared cache. Ubik [1], a Quality-of-Service (QoS) system for cache partitioning, provides tail latency guarantees for uniformly accessed caches. The Seesaw system attempts to implement Ubik and generalize it to non-uniformly accessed caches (NUCA) by applying its techniques to Jigsaw [2], a NUCA cache partitioning and data scheduling system. Early results suggest that under our current system, latency-critical applications are generally able to meet their deadlines, but an improvement in batch application throughput was not observed. This may be because the simulated system was not under enough load to require QoS.

2 Introduction

As the number of cores on a chip multiprocessor (CMP) increases, it becomes impractical to offer uniform access latency to a shared last-level cache (LLC) [3]. Newer systems spread the LLC banks across the cores, leading to a non-uniform cache access time, or NUCA (Figure 1). Current commercial systems are not NUCA-aware; data are typically striped across banks, leading to uniform—but mediocre—latency. Servicing cache misses, therefore, becomes a significant portion of overall program latency.

Figure 1: NUCA cache architecture



In modern OS runtimes, resources are allocated by some metric, generally maximization of utility. One limitation of this metric is that all applications are treated identically; however, some applications have stricter latency requirements than others (e.g. soft real-time vs batch). Running both on the same machine without some guarantee of quality of service (QoS) would significantly degrade performance; the latency-critical applications would have low utilization, so resources would be diverted to the batch processes. This prevents servers from efficiently running batch jobs and latency-critical jobs together, leading to low utilization. The StaticLC partitioning example in Figure 2 shows a static configuration where latency-critical applications are always allocated enough cache space to meet deadlines. Since these applications run infrequently, a lot of cache space which could otherwise be used by batch applications is wasted.

A complication common to all of these use cases is that the resources needed in excess of standard levels change at a fine granularity (10s of ms). For example, a latency-critical application may only need additional cache space near the end of a deadline. Allowing it to keep this space afterwards would impact the batch operation without helping the latency