Performance Isolation for Real-Time Applications on Multicore Platforms using PALLOC and MemGuard

Santosh Gondi, Siddhartha Biswas, Heechul Yun
The University of Kansas
2335 Irving Hill Road, Lawrence, KS
{santoshg, sid.biswas, heechul.yun}@ku.edu

Abstract

Performance isolation among multiple programs on a multicore platform is difficult to achieve due to contention in the shared architectural hardware resources such as shared caches and DRAM. In this paper, we present a case study—using WebRTC, an open source real-time video conferencing software developed by Google—that compares state-of-art techniques to achieve performance isolation. We first investigate the performance variability of WebRTC in the presence of memory intensive co-running applications. We then compare CPUSET (baseline), PALLOC (DRAM bank partitioning technique), and MemGuard (memory bandwidth reservation technique) in terms of performance isolation of the WebRTC application as well as the overall throughput of the entire system.

In this study, we found that PALLOC achieves moderate real-time performance improvements without significantly sacrificing the overall throughput. On the other hand, we found that MemGuard can be configured to achieve near perfect performance isolation for real-time applications, albeit at the cost of significantly reduced throughput for co-runners.

1 Introduction

Performance isolation is important for real-time applications such as real-time video conferencing software. On modern multi-core platforms, however, it is increasingly difficult to achieve because contention in shared hardware resources can cause high performance variations.

The CPUSET subsystem of Linux CGROUP is a well-known isolation mechanism which provides a basic core-level partitioning capability by confining applications in a given CGROUP partition to a subset of cores. Unfortunately, however, core level partitioning does not provide sufficient performance isolation when the shared memory resources—LLC capacity, DRAM banks, and memory bandwidth—become bottlenecks.

Recently, two new OS level performance isolation mechanisms, PALLOC [1] and MemGuard [2], have been proposed to provide better performance isolation for the shared memory resources on COTS multi-core platforms. PALLOC is a DRAM bank aware memory allocator, which enables us to partition banks among cores to avoid bank sharing. MemGuard is a kernel level memory bandwidth reservation system which provides a minimum bandwidth guarantee to each core.

In this study, we investigate the impact of these performance isolation mechanisms using a real world real-time video conferencing software WebRTC. WebRTC is an open-source, real-time video conferencing software developed by Google, which provides real-time video communication between browsers without having to install any plug-ins. Our goal is to provide a high degree of performance isolation to WebRTC on a commodity multi-core platform in a multi-programmed environment. We first investigate the performance variability of WebRTC in the presence of memory intensive co-running applications. We then compare CPUSET (baseline), PALLOC and MemGuard in terms of performance isolation of the WebRTC application as well as the overall throughput of the entire system.

With CPUSET mechanism alone, performance of WebRTC, indirectly measured via achieved memory bandwidth, is reduced by 27%, leading to 30% drop in the frame rate and 8 times increase in the RTT value. With PALLOC (in conjunction with the CPUSET), we achieve 10% increase in memory bandwidth, which leads to 30% improvement in RTT value compared to CPUSET isolation alone while experiencing only 15% performance reduction in the co-running LBMs. With MemGuard, we evaluated the performance impact of reservation-only, reclaim, and spare bandwidth strategies, and with different regulation period; we are able to come up with a configuration which completely eliminates the interference to WebRTC and Xserver from co-running applications.

In this study, we found that PALLOC achieves moderate real-time performance improvements without significantly sacrificing the overall throughput. On the other hand, we found that MemGuard can be configured to achieve near perfect performance isolation for real-time applications, albeit at the cost of a varying degree of reduced throughput for co-runners.

The remaining sections are organized as follows: Section 2 reviews the background on different isolation mechanisms on multicore platforms. Section 3 discusses the characteristics of WebRTC. Section 4 describes the experimental setup. Results and analysis are presented in Section 5. We discuss the limitations in Section 6. Paper concludes in Section 7.

2 Background

Most of the modern computers and hand-held devices are multi-core, multi-programmed systems. The applications running on these platforms experience a high degree of performance variation, because of contention for shared resources, such as LLC, memory banks, memory bandwidth, etc.

A general purpose multicore system runs a number of tasks at a time. If a memory intensive application is considered, its performance will depend not only on its own memory access pattern and system memory size and, but also with the memory access patterns of other co-running applications, and the state of DRAM. This problem is more obvious in real-time systems which can have only a small memory and that must be shared with all application running in that system. A similar situation arises for cache-intensive applications in multicore systems. In multicore systems, usually Level 3 cache is shared among different cores. So an application's performance will depend on both cache size and access pattern of shared cache by different processes.
These interference due to shared access of resources—memory, cache, and system bus—are critically important in designing real time systems which uses multicore processors. The following subsections review some of the performance isolation mechanisms for multi-core platforms.

2.1 CPUSET

Linux operating system provides a kernel level feature called, CGROUP (control group), to allocate system resources such as system memory, CPU cores, network bandwidth, or combinations of these resources to a user defined group of processes in the system. A user creates these groups following some user defined hierarchical structure to distribute the system resources among these groups so that the user can have proper control over managing system resources and to increase the performance of a selective group or overall performance improvement of the system.
Among several subsystems provided by control groups, CPUSET subsystem enables the user to assign individual CPU cores and memory nodes selectively to a particular group. For example, the parameter, cpuset.cpus, can be used to specify a CPU, which will be used by a particular task group, in control group virtual file system.

2.2 PALLOC

In multicore systems, DRAM is an important shared resource. Inside DRAM, there are multiple memory units, called banks, which can be accessed in parallel [1]. Each bank consists of multiple rows for storing data which must be accessed sequentially. Unfortunately, modern operating systems do not consider banks when allocating memory and considers whole DRAM as a single unit. As a result of this, data are stored in multiple banks in some scattered way. When multiple cores need to access these scattered data among multiple banks, performances of applications depend heavily on the location of these data. For example, if multiple cores executing different applications whose data is stored in same bank, must access that bank sequentially to fetch data, causing huge delay in the memory access, leading to degradation in the application's performance. Apart from this, memory controllers are also configured to interleave banks to improve bank level parallelism which makes the probability of sharing banks by different applications higher.

PALLOC is a DRAM bank aware memory allocator which is able to allocate memory to a particular bank. Using PALLOC, it is possible to partition DRAM Banks which can be assigned to individual cores, there-by reducing the interference caused by multiple core accessing same banks. It is also possible to achieve cache partitioning as side effect of bank partitioning, since the physical address bits determining memory banks and cache lines may share some bits. In most multicore platforms L3 cache is shared using same physical address. Because higher level cache is shared among multiple cores, interference occurs when different cores try to access higher level cache at the same time. In our case study, we collected the performance of WebRTC using PALLOC, various bank and shared cache partitioning configurations, and compared the performance with other mechanisms.

2.3 MemGuard

Apart from sharing the system DRAM and shared cache, memory bus is also an important shared resource. MemGuard is a memory bandwidth reservation system, which is able to provide guaranteed as well as best effort memory bandwidth service. The guaranteed bandwidth is the minimum reserved bandwidth ensured for each core, whereas, in the best effort mode, reclaiming and proportional sharing are supported. When reclaiming is enabled, an EWMA (estimated weighted moving average) function is used to predict the memory bandwidth requirement of each core, and its memory consumption limit is set to the estimated value. The additional bandwidth donated by a core is utilized by other cores as needed. This mechanism is useful with applications whose memory consumption pattern is bursty or does not follow any pattern. In the spare sharing mode, the bandwidth allocation period is restarted after each core consumes its allocated quota. This helps in improving the overall bandwidth utilization of the system.

3 Case Study

Real-time systems are generally classified as hard and soft real-time systems. Hard real-time systems have strict deadlines. Such as, nuclear fusion controller, robot performing surgery, etc. Missing such deadlines could be catastrophic. Whereas, in soft real-time systems, meeting the deadline is important, but missing the same is not that catastrophic. Video games, watching Netflix, video conferencing, etc. fall in this category. WebRTC, an open source, video conferencing solution, a soft real-time system, is considered for this case study. Because of resource contention on modern multi-core, multi user system, many soft real-time applications like WebRTC cannot meet the deadline in general purpose operating systems, such as, Linux. We first study the characteristics of WebRTC in conjunction with Xserver, by observing the memory access pattern in each case. Then, benefits and drawbacks of using PALLOC, and MemGuard are evaluated.

3.1 WebRTC

Figure 1 shows the number of cache-misses per ms, for one second duration. The mean memory bandwidth consumption is 342 MB/s, with peak value of 1247 MB/s. The bandwidth consumption is consistently bursty. The number of bursts in one second correlates to the video frame rate of 30 FPS.

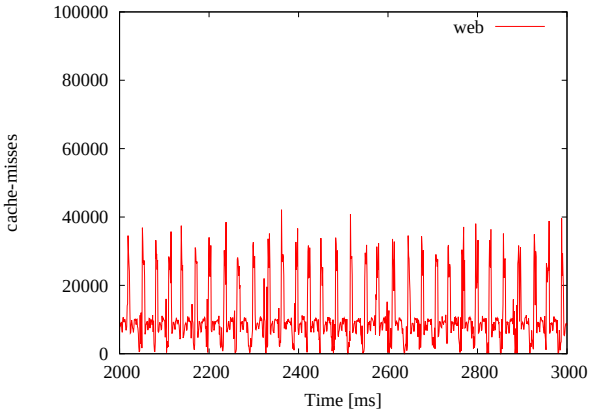

FIGURE 1: *Memory access pattern of WebRTC application*

3.2 X11

Similar to WebRTC, Xserver exhibits burst memory consumption as shown in the figure 2. Since Xserver is the rendering front end of WebRTC streams, it sort of follows the same pattern as WebRTC. There is aggregation of 2 memory spikes in Xserver compared to WebRTC, probably because of the way two processes interact. The mean memory consumption of Xserver is 365 MB/s, with the peak value of 2865

MB/s.

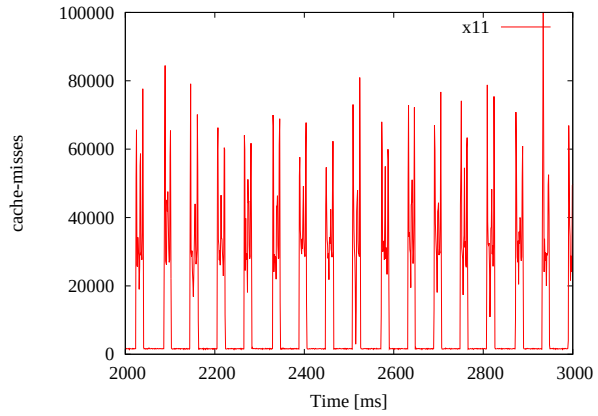

FIGURE 2: *Memory access pattern of X11*

Resource reservation based on the mean requirements will not be able to provide sufficient isolation from co running application during bursty memory consumption. However, the reservation based on peak requirements will not be able to utilize the spare memory bandwidth during non-burst phases. Isolation mechanisms which dynamically reserves the memory based on the run-time requirement is more suitable for bursty memory consumers. By understanding the memory access pattern of WebRTC and X11, we are able to configure the isolation mechanisms to achieve near perfect performance isolation from other co-running tasks. In this case study, we have explored the CPUSET, PALLOC, and MemGuard mechanisms to provide performance isolation to WebRTC.

4 Experiment Setup

Figure 3 shows the experimental setup. Both machines in the experiment have identical configuration. Each machine is equipped with a quad core Intel Xeon processor, 2.8 GHz CPU, 8MB 16-way shared L3 cache, and 256KB private L2 cache. Each platform has 4 GB DDR DIMM memory with 16 banks, and an integrated memory controller. The WebRTC and Xserver performance, and effectiveness of isolation mechanisms is measured on one machine, while, the browser on other machine simply serves as the other end point of WebRTC.

In each experiment, 3 Linux cgroups are created, namely, WebRTC, Xserver, and CoRun. Cpu cores 0,1,2-3 are assigned to WebRTC, Xserver, and CoRun control groups, respectively. The chrome-browser running WebRTC is assigned to WebRTC group, Xserver process is assigned to Xserver group, and lbm benchmarks from SPEC2006 are assigned to CoRun group. Memory bank partitioning is achieved by assigning banks 0-3 to WebRTC, 4-7 to Xserver, and 8-15 CoRun groups. Similarly, bank and cache partitioning is achieved by assigning 0 to WebRTC, 1 to Xserver, and 2-3 to CoRun groups. Since there are 2 bits shared between bank and cache selection bits on address line [1], there are only 4 available partitions for bank+cache partitioning.

‘ To consistently repeat the experiments, a v4l2loopback device is used as a virtual camera, and foreman YUV sequence is used to generate a repeatable sequence of video frames [4]. The Java script code in application is modified to support 4 MB/s bandwidth in both directions, instead of default 2 MB/s.

The two hosts in the experiment are connected through a dedicated gigabit switch. After initial handshake through a local signaling server, there is direct point to point connection between the two communicating hosts while the WebRTC session is going on. The end to end RTT is of the order of 2 ms. Since there is no other cross traffic through the switch, the performance degradation observed in WebRTC is completely because of resource contention in the end hosts, rather than the network itself.

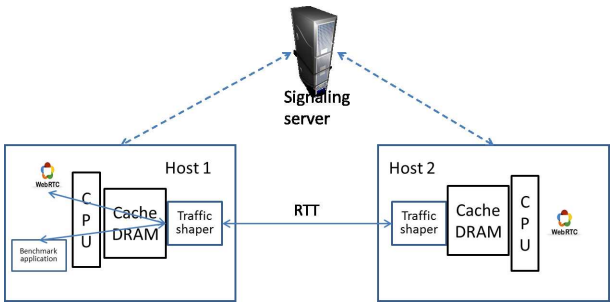

FIGURE 3: *Experiment setup*

5 Results and Analysis

In this section we present the results and analysis of different performance isolation mechanisms.

5.1 CPUSET and PALLOC results

Figures 4 to 8 shows the memory bandwidth achieved by WebRTC, Xserver, and co-running LBM tasks in different settings. Each value is an average of 3 iterations. Figures 4 and 5 shows the baseline performance with cpuset, bank, and bank+cache parti-

tioning. Abbreviations, CPUSET, PB, and PB+PC respectively correspond to these schemes. With cpu core partitioning alone, The Xserver's memory bandwidth drops to 92 MB/s from 365 MB/s in solo run. At the same time, WebRTC's bandwidth drops to 250 MB/s from 342 MB/s in solo run. Two co-running LBM tasks achieve close to 1500 MB/s bandwidth as shown in figure 5. With bank partitioning (PB), Xserver WebRTC gain around 20 MB/s more bandwidth, albeit, the co-running lbm tasks lose around 200 MB/s of bandwidth. This result shows that, though bank partitioning is able to resolve some bank conflicts, it is not sufficient to provide complete isolation to WebRTC and Xserver, as major contention seems to be arising from memory bus bandwidth. The drop in co-runner's performance is because of constraining memory allocation to these tasks to specific banks, hence reducing the overall availability of memory. With bank+cache partitioning (PB+PC), WebRTC performance improves marginally as the IPC increases, while the bandwidth reduces, indicating that some of its cache conflicts got eased. But, it does not provide any additional gains for WebRTC and Xserver combined. Also, it further reduces the performance of co-running applications by around 70 MB/s. The result confirms that, cache is not the main contentious resource on this platform, rather, the bus bandwidth is. The further reduction in co-running lbm's performance is because of constrained cache size leading to increased cache evictions.

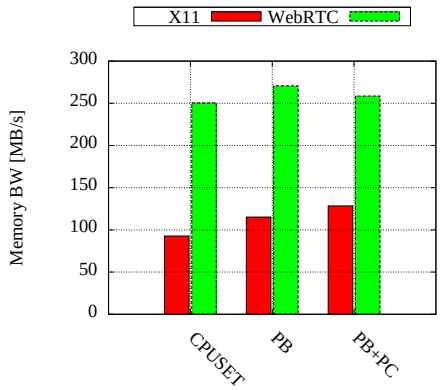

FIGURE 4: *Memory BW of X11 and WebRTC with cpuset, PALLOC, and cache partition*

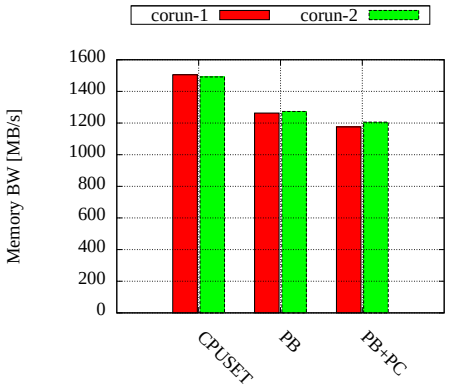

FIGURE 5: *Memory BW of Co-runners with cpuset, PALLOC, and cache partition*

5.2 MemGuard results

Figures 6 to 8 shows the achieved memory bandwidth for different MemGuard settings. The abbreviations: MG-RO, MG-PS, MG-BR, and MG-BR-PS correspond to MemGuard with reservation only, proportional sharing, bandwidth reclaiming, and bandwidth reclaiming with proportional sharing. Figures 6 to 7 are with 1 ms regulation period and figure 8 is for 10 ms regulation period. With allocation of 600, 600, 100, and 100 MB/s to cores 0-3 (Figure 6), WebRTC and Xserver get the sufficient share of bandwidth. Because, both WebRTC and Xserver are bursty and LBM are memory intensive, bandwidth reclaim increases the LBM performance beyond the allocation quota for co runner applications, by reclaiming underutilized bandwidth from Xserver and WebRTC in many bandwidth regulation periods. The highest performance is observed for with both reclaim and proportional sharing, because of possibility of shortened regulation periods due to proportional sharing.

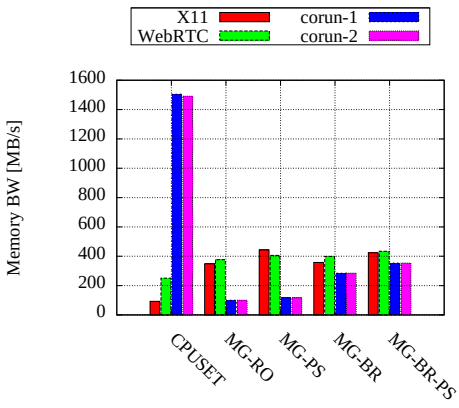

FIGURE 6: *MemGuard with 600 600 100 100 allocation and 1 ms period*

With allocation of 400, 400, 300, and 300 MB/s to cores 0-3 (Figure 7), the gains for co-runner due to bandwidth reclaiming is minimal since the difference in allocation is small. The combined gains of reclaiming and proportional sharing is high, because, reclaim helps to complete the bandwidth quota for each task quickly, and underutilized regulation periods are shortened because of proportional sharing. Overall the bandwidth of WebRTC and Xserver is lower because of possibility of memory controller queue being congested, due to more aggressive memory consumption by lbm co-runners.

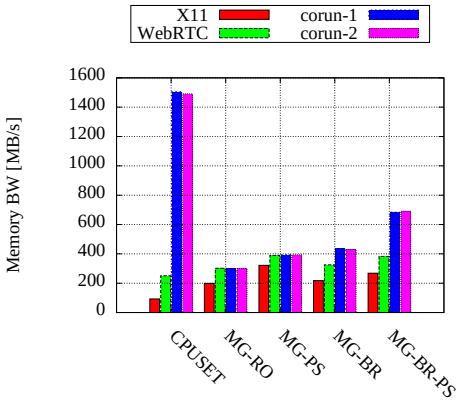

FIGURE 7: *MemGuard with 400 400 300 300 allocation and 1 ms period*

Figure 8 shows the results with 10 ms regulation period of MemGuard. In all cases, bandwidth achieved by co-running applications is same or higher compared to 1 ms period. With reclaim and proportional sharing, the higher memory consumption by co-runners causes unwanted side effect of slightly reduced bandwidth for WebRTC and Xserver. Again, this is possibly because of increased queuing at memory controller. We are currently investigating this issue to determine the exact cause of this behavior. The overall throughput is higher because of the reduced overhead of regulation and more time for aggregation of memory consumption. It should be noted that higher regulation period has negative side effect of increasing response time for applications.

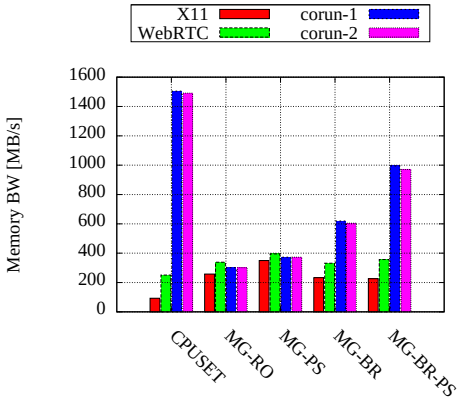

FIGURE 8: *MemGuard with 400 400 300 300 allocation and 10 ms period*

5.3 WebRTC performance

Table 1 shows the average RTT, frame rate, and sent media bandwidth of WebRTC, while running alongside the co-runners, for different isolation mechanisms discussed in this paper. With CPUSET partitioning alone the RTT increases by 8 times compared to solo run and frame-rate and bandwidth reduces by more than 30%. With bank (PB) and bank+cache (PB+PC) partitioning, the RTT and media bandwidth improves by 20%. MemGuard (MG) configuration achieves nearly perfect isolation by producing RTT, frame-rate, and bandwidth figures similar to the solo run. WebRTC utilizes GCC (google congestion control) [4] algorithm to dynamically adjust the encoding bit rate to the available link bandwidth. Since, in our experimental setup there is no network congestion between two WebRTC hosts, the video quality variation, quantified by frame rate and media bandwidth together, is solely determined by the resource contention on the end host.

	RTT(ms)	FR(f/s)	BW(KB/s)
CPUSET	17.20	21.34	2917.85
PB	12.62	21.34	3407.67
PB+PC	11.28	20.85	3594.00
MG	2.24	29.98	4019.16
Solo	1.85	30.00	4022.33

TABLE 1: *Average RTT, Frame-rate, and Bandwidth*

6 Discussion

Average bandwidth and bandwidth access pattern are important in determining the appropriate isola-

tion mechanism, in-order to provide perfect performance isolation for the real-time applications. In our case study, we could find the memory access pattern of WebRTC and X11 tasks using system logs from MemGuard module. This has helped us in appropriately configuring the MemGuard to achieve our objectives. However, it may be difficult to dynamically determine the memory requirements of an application so as to use it as input to the memory reservation system. We plan to work on building a mechanism for applications to specify the memory requirements, and use it for memory bandwidth reservation.
Also, in our case study, the X11 does not use hardware acceleration. Since the present day devices in all form factors are using GPU acceleration, we would like to evaluate the effectiveness and side effects of different performance isolation schemes, with GPU acceleration enabled.

7 Conclusions

Determinism is cornerstone of real-time systems. But, contention for shared resources on multi-core systems makes it difficult to achieve deterministic performance. In our case study, we verified the effects of resource contention on WebRTC on a general purpose multi-core platform. The basic CPU core partitioning is not sufficient, because, the LL3 cache, memory bus, and memory banks are still shared by the cores. Bank, and cache level partitioning improve the performance of WebRTC to some extent, without significantly harming the co-runner performance. Since the WebRTC (and hence the X11) memory demand is bursty, a strong resource reservation scheme was necessary to completely isolate the real-time performance. But, strict reservation had a side effect of significantly reducing the performance of co-runner application. By understanding the memory access pattern of WebRTC, we were able to device a configuration of MemGuard with suitable memory bandwidth division, spare bandwidth utilization and reclaim policy, to achieve the best real-time performance and higher system throughput.

References

[1] *PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms*, Heechul Yun, Renato, Zheng-Pei Wu, Rodolfo Pellizzoni, 2014,IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)

[2] *MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms*, Heechul Yun, Yao Gang, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha, April 2013,Real-Time and Embedded Technology and Applications Symposium (RTAS)

[3] *http://www.webrtc.org/*

[4] *Experimental Investigation of the Google Congestion Control for Real-Time Flows*, Luca De Cicco, Gaetano Carlucci, Saverio Mascolo, 2013,FhMN '13 Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking