Optimizing Google's Warehouse Scale Computers: The NUMA Experience

Lingjia Tang[†], Jason Mars[†], Xiao Zhang[‡], Robert Hagmann[‡], Robert Hundt[‡], and Eric Tune[‡]

[†]University of California, San Diego
{lingjia, mars}@cs.ucsd.edu

[‡]Google
{xiaozhang, rhagmann, rhundt, etune}@google.com

Abstract

Due to the complexity and the massive scale of modern warehouse scale computers *(WSCs), it is challenging to quantify the performance impact of individual microarchitectural properties and the potential optimization benefits in the production environment. As a result of these challenges, there is currently a lack of understanding of the microarchitecture-workload interaction, leaving potentially significant performance on the table.*

This paper argues for a two-phase performance analysis methodology for optimizing WSCs that combines both an in-production investigation and an experimental load-testing study. To demonstrate the effectiveness of this two-phase approach, and to illustrate the challenges, methodologies and opportunities in optimizing modern WSCs, this paper investigates the impact of non-uniform memory access (NUMA) for several Google's key web-service workloads in large-scale production WSCs. Leveraging a newly-designed metric and continuous large-scale profiling in live datacenters, our production analysis demonstrates that NUMA has a significant impact (10-20%) on two important web-services: Gmail backend and web-search frontend. Our carefully designed load-test further reveals surprising tradeoffs between optimizing for NUMA performance and reducing cache contention.

1 Introduction

As much of the world's computation continues to move into the cloud, the computing demand on the class of datacenters recently coined as "warehouse scale computers" (WSCs) [12] rises. As such, it becomes increasingly important that the performance and utilization of the machines housed in WSCs are maximized. However, inefficiencies and missed performance opportunities remain rampant in modern WSCs. Although Google has architected and deployed one of the largest and most advanced datacenter infrastructures in the world, we observe it is not free from inefficiencies and missed opportunities.

At the high level, this problem stems from a lack of understanding of the interaction between our web-service workloads and the underlying machine architectures housed in our WSCs. This outcome is the natural result of the design philosophy of abstracting the underlying computing resources in WSCs as homogeneous across various idiosyncratic microarchitectural and system designs. Concretely, a WSC is viewed as a collection of thousands of cores, terabytes of main memory, petabytes of disk space, etc., without an explicit notion of microarchitectural resources and features such as on-chip caches, non-uniform memory access, off-chip bandwidth, etc. Evolving WSC designs to recognize and optimize for these microarchitectural properties may provide large performance and utilization advantages. However, investigating the opportunity cost of optimizing for these microarchitectural properties in production is a challenge in itself, and methodologies to accomplish this type of analysis are few.

[Challenges] Ideally, we aim to study the interaction between workloads and architecture as it relates to the scheduling of jobs to machines, and the individual threads of a job to the cores across sockets of a given machine. We are interested in studying the interaction in the production environment. However, unlike the benchmarking or load-testing of individual applications in a controlled environment, it is difficult to link microarchitectural properties or effects to individual applications or execution scenarios in production for a number of reasons.

1. At the scale and complexity of production execution,

978-1-4673-5587-2/13/$31.00

performance factors are intertwined. When collecting performance measurements in production, it is the interaction of these factors that are captured, making it quite challenging to investigate the performance impact of each individual factor.

2. In addition, all of the factors impacting a performance measurement may not be known, and can change spontaneously, such as load or user behavior.

3. Further exacerbating this challenge is the fact that sufficient but low-overhead instrumentation to characterize and collect various performance effects must be designed and deployed in the WSC to adequately diagnose and decompose these effects.

In practice, production measurements are often noisy with large and sometimes seemingly inexplicable performance swings. The compounding factors responsible for these swings make it difficult to reason about individual effects on individual applications. This inability to isolate factors for scrutiny is one of the major challenges facing WSC architects. For example, upon investigating the performance of numerous instances of Gmail's backend server, we observe around a 4x range in average request latency during a week's time. Note that, for this measurement, we only collect performance information for instances of Gmail running across a number of identically configured machines in a single cluster with similar user activity profiles. It is overwhelmingly challenging to diagnose and attribute this performance swing to individual microarchitectural factors. Effects such as the contention for cache/bandwidth with various corunning applications on a server, non-uniform memory accesses (NUMA), and I/O interference among other factors all carry implications on the effectiveness of the policies used for cluster-level scheduling, machine-level resource management, and the execution configurations of the Gmail servers. Other factors not explicitly measured in this particular Gmail study, such as load imbalance and fluctuation, user accounts migrations, fault tolerance mechanisms (master and slave copy flips), and even datacenter maintenance, may also be responsible for these large performance swings, further complicating the investigation. This example illustrates that, when investigating performance effects and anomalies in a production environment, it is difficult to tease out individual factors and study their respective performance impacts. As a result of the challenge of estimating the benefit a particular change or optimization will produce, architects and engineers may resort to simply implementing and deploying a solution before fully understanding the benefit. The downside to this approach is that implementation and deployment are costly, and only the clearly promising ideas can be entertained.

[Contributions] In this work, we argue for a two-part performance analysis and characterization methodology for investigations at the WSC scale. We illustrate the opportunities, challenges, methodologies and future directions in identifying and architecting for performance opportunities in modern WSCs by conducting an investigation of how several widely used services in Google's WSCs are affected by NUMA. Our methodology combines the following two parts:

- An in-production cluster-level investigation to quantify the potential benefit of managing NUMA. We designed a novel metric for NUMA locality that allows us to deploy a low-overhead large-scale profiling in production. We then relied on statistical methods to analyze the noisy production profiles to correlate NUMA behaviors with the responsible performance swings, teasing out the impact of other factors.

- An experimental load-testing approach at the single-server level to gain a more in-depth understanding of application level characteristics and the interaction between NUMA and other microarchitectural properties. While the production analysis sheds light on the performance opportunities in a real-world setting with the existence of various other compounding performance factors, the single-node load-testing allows us to further establish the distinction between correlation and causality to hone in on the performance effects as well as the interaction between NUMA and other performance factors.

This paper focuses on a potentially significant opportunity for improving performance and efficiency in WSCs, namely, the explicit consideration of non-uniform memory access (NUMA) when executing jobs in WSCs. Understanding the impact of NUMA on large-scale commercial web-service workloads provides a potentially significant performance opportunity and is critical as NUMA designs are dominant in today's server markets. There has been a wealth of prior research regarding NUMA-related scheduling approaches [1, 4, 5, 9, 18–20, 27, 28]. However, in contrast to NUMA on a single machine, or the treatment of a cluster of distributed shared memory seen as a single NUMA computing unit, new implications are introduced at the scale of thousands of NUMA machines constituting a single WSC. Due to the challenge of characterizing the interaction between an individual microarchitectural property and application performance in production WSCs, the impact of NUMA on large-scale datacenter applications remains unknown. And at the scale of modern WSCs, it remains unclear whether the underlying system platforms are effectively managing NUMA resources to maximize performance and utilization.

[Results] Our production analysis results demonstrate that a sizable performance opportunity exist in large-scale WSCs as NUMA has a significant performance impact on

datacenter applications in production (10-20%). Our load-test confirms the production results in general. Moreover, the load-test experiments reveal surprising results that in some scenarios, more remote memory accesses can outperform more local accesses significantly (by up to 12%) due to the interaction and tradeoffs between NUMA locality and cache sharing. To the best of our knowledge, this is the first work to investigate the interaction between NUMA locality and cache sharing/contention using large-scale datacenter applications in production environment and demonstrates some counter-intuitive insights.

The rest of the paper is organized as follows. Section 2 presents our study in production WSCs. Section 3 presents our study using single node load-test. Section 4 discusses our methodology and gained insights. Section 5 presents related work and Section 6 concludes.

2 Impact of NUMA: Production Cluster-Level Study

This section presents our cluster-level study in production WSCs to quantify the impact of NUMA on large-scale datacenter applications. We first design a metric to quantify the amount and the average distance of remote accesses for a job (a running instance of an application), namely, the *CPU-memory locality score* or *NUMA score*. This simple metric allows us to perform lightweight and continuous profiling of the CPU-memory locality for all jobs in a large-scale production environment. In addition to the locality score, we also monitor and sample the performance of each job, including its cycles per instruction (CPI) and its application-specific performance metrics such as request latency. Lastly, we conduct correlation analysis based on the collected production samples to investigate the relationship between CPU-memory locality and the application performance to quantify the performance impact of NUMA.

2.1 Quantifying CPU-Memory Locality

To quantify the CPU-memory locality of an individual job, we must capture its runtime CPU and memory usage across different NUMA nodes on a machine. An example of such NUMA machine is shown in Figure 1. To capture the CPU usage of a job, we simply aggregate the OS exported per-CPU usage statistics for each job. To calculate per-node memory usage for each job, ideally we want to count memory accesses among all NUMA nodes using hardware performance counters. However, it is challenging to break down those counters on a per-CPU basis (i.e. some counters are counting for a group of CPUs that are siblings in the same domain) and accurately attribute them to concurrently running jobs on the same machine. Therefore, we use each job's allocated memory page numbers and locations instead, which are exported by the kernel, to approximate per-node memory usage. The advantage of this approximation is that it is lightweight for online profiling with low overhead. We then normalize per-node CPU-memory usage such that they sum up to 1 over all NUMA nodes. Let $C[1..n]$ and $M[1..n]$ respectively denote normalized per-node CPU and memory usage for a job on a n-node machine, and $D(i,j)$ denotes distance between two nodes i and j. The *CPU-memory locality score* (or *NUMA score*) of a job can be calculated as below:

$$Score = \sum_{i=1}^{n} \sum_{j=1}^{n} C[i] \cdot M[j] \cdot \frac{D(i,i)}{D(i,j)} \quad (1)$$

The node distance $D(i,j)$ is a machine-dependent table which can be populated before hand. In our evaluation, we use ACPI (Advanced Configuration and Power Innterface) defined NUMA distance [17] for $D(i,j)$. Specifically, a node to itself has distance 10, 1-hop away node pair has distance 20, 2-hop away node pair has distance 30. The locality score is between 0 and 1. We deploy the profiling mechanism in production to periodically update and store the CPU-memory locality score of all jobs.

2.2 Impact of NUMA in Production

Leveraging the locality score as described earlier, we conduct our study using two large-scale applications: `Gmail backend server` and `websearch frontend` in production. Both are important applications in our WSCs. For example, Gmail is one of the biggest email services in the world. It is also one of the top web-services that consume an enormous amount of resources in our production WSCs. Improving its back-end server performance is critical for improving user experience and reducing cost in production. In addition to profiling the CPU-memory locality score (Equation 1) of each `Gmail backend server` job, we also sample the performance of each job for our correlation analysis. As we mentioned earlier, one challenge for studying the performance impact of NUMA in production is the existence and influence of other performance factors in the complex production environment, such as heterogeneous machines, user migration, datacenter maintenance, uneven and fluctuating load, as well as co-location with other applications. For example, `Gmail backend server` jobs are run in various datacenters across the globe on several types of machine platforms that are from different vendors and different generations. The load for Gmail constantly fluctuates. Although user accounts migration among machines and datacenters across the globe is conducted regularly for load balancing, the load still may not be evenly distributed across machines within a cluster or across datacenters. In addition, `Gmail backend server` jobs are not run in dedicated clusters.

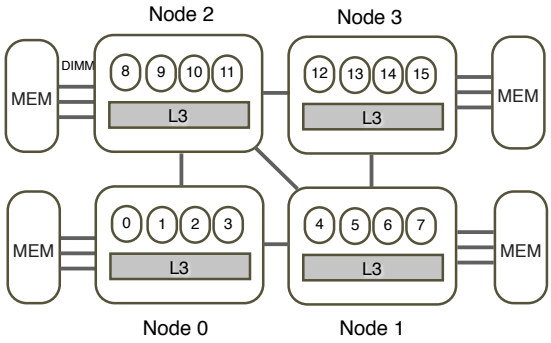

Figure 1. AMD Barcelona

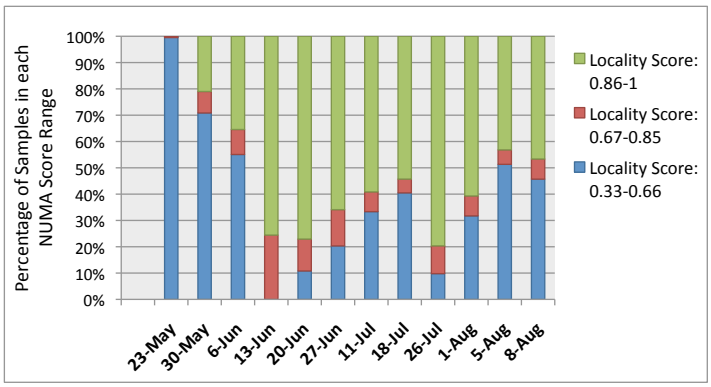

Figure 2. Percentage of `Gmail backend server` jobs within various locality score ranges.

Therefore, each job may be co-located with other applications on a multicore server. To minimize the influence of these factors, we collect a large amount of samples of hundreds of Gmail server jobs at a fine granularity every day for a few months from identically-configured AMD Barcelona platforms in one production cluster. This AMD Barcelona platform is shown in Figure 1, on which, four nodes are asymmetrically connected using bi-directional HyperTransport.

[NUMA Score Distribution] We first investigate the percentage of `Gmail backend server` jobs that are having remote memory accesses. Figure 2 presents the distribution of jobs in different CPU-memory locality score (NUMA score) ranges. The locality score of each job on each machine is sampled periodically (every 5 mins) on AMD Barcelona in one production cluster. This figure summarizes the sampling results for every Monday in a three-month span (23/05 to 08/08). Each day, around 65k samples are collected. On our AMD Barcelona platforms (Figure 1), the NUMA score ranges from 0.33 (all memory accesses are 2-hops away) to 1 (all accesses are in the local memory node). Figure 2 shows that NUMA score distribution fluctuates. On May 23rd, all jobs are having 100% remote accesses. The situation improves until June 13rd, when all samples have locality score higher than 0.66, and then it deteriorates again. This fluctuation may be due to job restarts, machine restarts, kernel updates, other high priority jobs get scheduled to the machines, etc. But in general, on average, for a significant amount (often more than 50%) of jobs, all memory accesses are at least 1 hop away.

[Correlating NUMA Score and CPI] To investigate if better memory locality necessarily indicates better performance and to quantify the amount of the performance swing due to local/remote memory accesses, we samples the `Gmail backend server job`'s cycles per instructions (CPI) and correlate a job's CPI with its locality score. We use CPI for as our performance metric because 1) we observe that in our production clusters, most latency-sensitive applications' average CPI measurements are fairly consistent across tasks and are well correlated with the application-level behavior and performance; 2) it can be sampled with very little overhead in production. Figures 3 and 4 present the results of the correlation analysis of all samples collected on two randomly selected Mondays. The x-axis is the NUMA score bin, and the y-axis shows the average normalized CPI of all samples that belong to that NUMA score range. The error bars show the standard deviations. These two figures show that the impact of NUMA on performance (CPI) of `Gmail backend server` is quite significant. In Figure 3, the CPI is dropping from around 1.15x (at score 0.5) to around 1x (at score 1). It is a 15% difference. Although the standard deviation is not small due to the influence of other performance factors, the trend is clear. In Figure 4, the average normalized CPI drops from 1.13x (at score 0.5) to 1x (score 1), a 14% reduction. This shows that for `Gmail backend`, in general, the more local accesses (higher locality score), the better the CPI performance (lower CPI). In addition to Gmail on AMD Barcelona, we also conducted similar study for `Web-search frontend` on Intel Westmere servers. The results are presented in Figure 5. Our Intel Westmere platform is a dual-socket Xeon X5660 NUMA machine, shown in Figure 9. Each chip has its own integrated memory controller and buses connecting to memory. Processors are connected through Intel QuickPath interconnect (QPI). Similar to Gmail, we observe a significant performance improvement when CPU-memory locality improves: on average the CPI drops from 1.20x to 1x when the locality score improves from 0.5 to 1.

[Correlating NUMA Score and Application Performance] In addition to CPI, we also correlate each `Gmail backend server` job's NUMA score with its user-

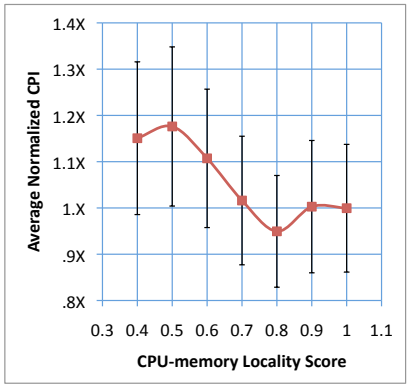

Figure 3. Gmail backend server on 05/30. Better NUMA score correlates with lower CPI.

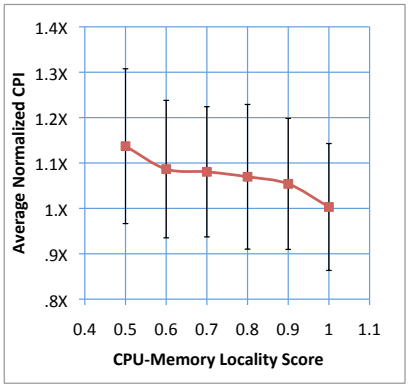

Figure 4. Gmail backend server on 06/20 - NUMA score and CPI.

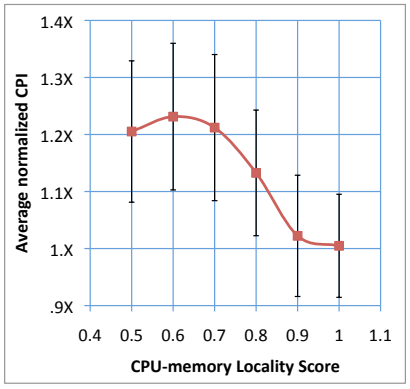

Figure 5. Web-search Frontend - NUMA score and CPI.

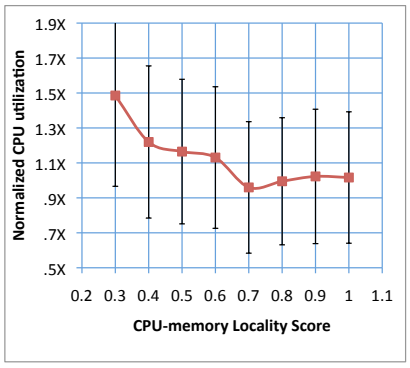

Figure 6. Gmail backend server. Better NUMA score correlates with better CPU utilization

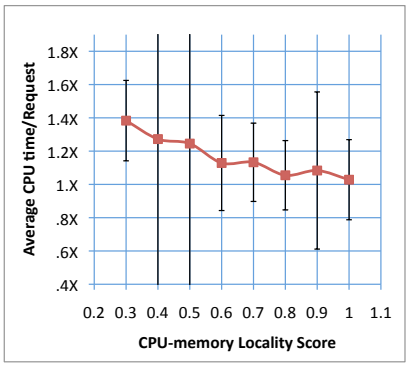

Figure 7. Gmail backend server's CPU time/request and NUMA score

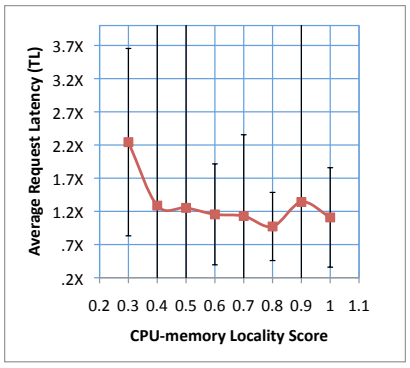

Figure 8. Gmail backend server's request latency (loading threadlist request) and NUMA score

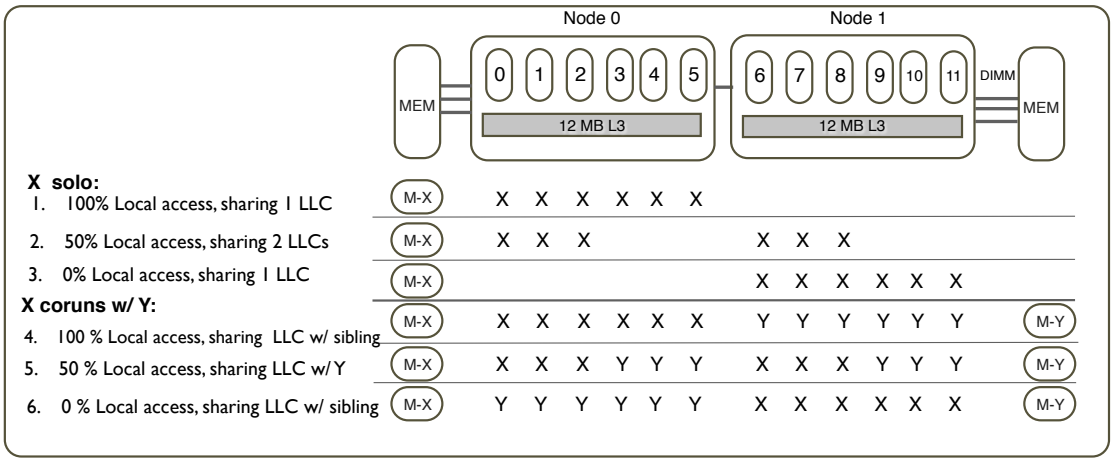

Figure 9. Intel Westmere and the 6 running scenarios in our load-test experiments

specified performance metrics. Three important performance metrics for `Gmail backend server` that we investigated are CPU utilization, CPU time/request and request latency (focusing on user requests to list email threads).

Figures 6 - 8 present the average performance of all jobs at each NUMA score range. The error bars show the standard deviations. The samples are collected on one Monday for all Gmail backend server jobs running on all AMD Barcelona in the same production cluster. CPU utilization is sampled every 1 minute. CPU time/request and request latency are sampled every 2 minutes. Figure 6 shows that CPU utilization improves when the amount of local memory accesses increases. The normalized CPU utilization drops from 1.5x (at score 0.3) scenario to 1.07x (at score 1), indicating 40% more CPU time is required for the same amount of work when most memory accesses are remote (score 0.3). Figures 7 and 8 also show similar correlations but the data are less conclusive because of the measurement noise. The standard deviations are significant especially when the locality score is in [0.4, 0.6) and [0.9-1) ranges. This may be due to the fact that user requests latency often has a significant performance swing during a single day even within the same cluster because of other performance factors including load changes.

[Summary] Our experiment results demonstrate that remote memory accesses have a significant impact (10-20%) on the performance (CPI and CPU utilization) of two large-scale applications (`Gmail backend` and `Web-search frontend` in production WSCs. However, due to the influence of other performance factors such as load fluctuation, our results for two other performance metrics (request latency and CPU efficiency) are fairly noisy and less conclusive. In the next section, we will conduct further investigations using single node load-test. More discussion of the results is presented after the load-test experiments in Section 4.

3 Impact of NUMA: Single Node Load Test

To further confirm and quantify the NUMA impact observed in production, we conducted single-node load-tests for several key Google applications. With carefully designed controlled experiments, the load-test also allows us investigate at a fine granularity how NUMA locality interacts with other microarchitectural factors on a commodity server. Such interaction study is very challenging, if not intractable, on the production cluster level.

One interesting interaction this work focuses on is the **tradeoffs between memory access locality and the impact of cache sharing/contention** on a CMP machine, including when multiple applications are running on the same machine simultaneously. Prior work [33] has recently demonstrated the significant performance impact from sharing memory subsystems such as last level caches and memory bandwidth on multicore machines. For large-scale datacenter applications, there can be both a sizable performance benefit from properly sharing caches to facilitate data sharing, and a potentially significant performance degradation from improperly sharing and contending for caches and memory bandwidth. However, neither recent work on cache contention or prior work on NUMA takes the interaction between memory locality and cache contention/sharing into consideration. The tradeoffs between optimizing NUMA performance by clustering threads close to the memory nodes to increase the amount of local accesses and optimizing for cache performance by spreading threads to reduce the cache contention remain unclear.

Table 1 presents the applications studied in our load-test as well as their performance metrics. A load generator is set up to feed the queries, collected from production WSCs, to these applications. The performance presented in this section is the peak load performance of each application after the initialization phase, and the performance is fairly consistent (within 1% difference) between runs. Our experimental platform is the same Intel Westmere used in the previous section shown in Figure 9.

In our study, we conducted both **solo** and **co-run** experiments. Six running scenarios (solo scenarios 1-3 and co-run scenarios 4-6), shown in Figure 9, are evaluated for each application. Each application is configured to have 6 threads. In this figure, "X" denotes the threads of the application of interest, "M-X" denotes where the memory of X is allocated. In the co-run scenarios, "Y" denotes the co-running application. In scenario 1, 100% of the "X" memory accesses are local. In scenario 3, 100% of the memory accesses are remote. Therefore the performance difference between these two scenarios for an application demonstrates the impact of NUMA when the application is running alone. In scenario 2, 50% of the memory accesses are remote. Also in this scenario, 6 threads of "X" are spread across 2 sockets using 2 last level caches (LLCs) instead of clustering to 1 shared cache as in scenarios 1 and 2. Scenarios 4-6 are similar to scenarios 1-3 except that application "Y" now occupies the rest of the cores.

Figures 10 - 12 present our results. In each figure, the y-axis presents the performance for each Google application in 6 running scenarios. The performance is normalized by the application's performance in scenario 1. The x-axis presents whether the application is running alone (solo), and when it is not, its co-running applications.

[Solo] The first cluster of three bars in each graph shows each application's solo performance in scenarios 1-3. The difference between the 1st and the 3rd bars in this cluster demonstrates the performance impact of NUMA. In general, each application is affected by the increase of re-

workload	description	metric
cluster-docs	Unsupervised Bayesian clustering tool to take keywords or text documents and "explain" them with meaningful clusters.	throughput
bigtable	storage software for massive amount of data	average r/w latency
search-frontend-render	Web search frontend server, collect results from many backends and assembles html for user.	user time (secs)

Table 1. Datacenter Applications

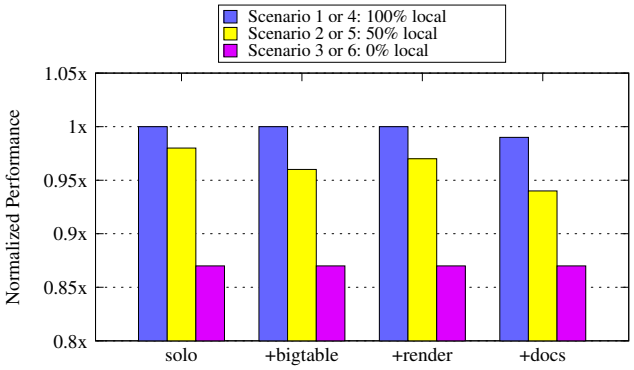

Figure 10. Normalized performance of `cluster-docs` running alone and running with 3 various co-runners

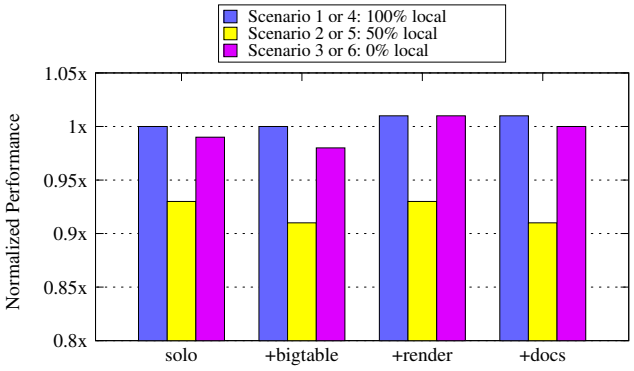

Figure 11. Normalized performance of `Bigtable`

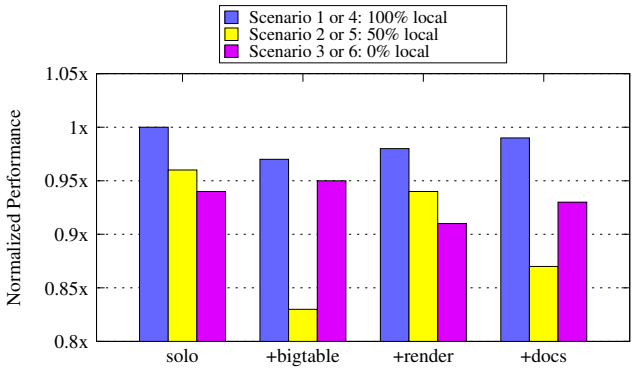

Figure 12. Normalized performance of `Web-search frontend render`

mote accesses. For example, `cluster-docs` has a 12% performance degradation when all accesses are remote, as shown by the first cluster of bars in Figure 10. However, `Bigtable` (Figure 11) stands out to be a curious case, whose performance for 100% remote accesses is better than 50% remote accesses. This may be due to the fact that `Bigtable` benefits from sharing cache among its own sibling threads. As reported in recent work [33] that Bigtable has a large amount of data sharing (confirmed by its performance counter profiles) and thus its performance benefits from sharing the last level cache among its threads. Thus when it cannot have 100% local accesses, interestingly, it may prefer to cluster its threads to a remote node (1 shared cache) than spreading them across sockets (2 shared caches) for partial local accesses.

Another interesting case is `cluster-docs` (Figure 10), whose performance degradation for 50% local accesses comparing to 100% local accesses is quite insignificant (1-2%). However, the 0% local accesses case has a significant performance impact. This is because that `cluster-docs`' threads contend for cache space [33]. Therefore, although spreading its threads (scenario 2) increases remote accesses, it also increases cache space, alleviating the performance degradation.

[Corun] The 2nd-4th clusters of bars in each figure demonstrate each application's performance in scenario 4-6

with 3 different corunners. Each cluster presents a different corunner shown as the x-axis. For `cluster docs` (Figure 10) and `bigtable` (Figure 11), the application performance is similar when running with other applications (the 2nd to the 4th clusters of bars) as when running alone (the 1st cluster of bars). Figure 12 shows that `search-frontend-render`*'s best running scenario changes as the co-runner changes*. This indicates that the tradeoff between NUMA and cache contention for `search-frontend-render` depends on which other applicatoin(s) it is running with. When running alone or running with `frontend-render`, it prefers to maximize the amount of local accesses. However, when running with `bigtable` or `cluster-docs`, it prefers scenario 5 (running on a different socket from its corunner) to scenario 6 (sharing caches with corunner), despite the fact that scenario 5 provides 50% local accesses rather than 0% in scenario 6. In this case, cache contention causes more performance degradation than 50% remote accesses. Note that in previous section, our analysis indicates that the performance of `search-frontend` in production is correlated with the NUMA score, indicating the more local accesses the better performance. And interestingly, our load-test shows that `search-frontend-render`, a component of `search-frontend`, sometimes may prefer remote accesses to cache contention.

In summary, our load-test experiments confirm that in general the more local memory accesses, the better the performance (up to 15% performance swing due to NUMA). However, surprisingly, the load-test also demonstrates that this conclusion does not always hold true. This is due to the interaction between NUMA and cache sharing/contention.

4 Summary and Discussion

[Results Summary and Insights] Both our production WSC analysis and load-test experiments show that **the performance impact of NUMA is significant for large scale web-service applications on modern multicore servers**. In our study, the performance swing due to NUMA is up to 15% on AMD Barcelona for Gmail backend and 20% on Intel Westmere for Web-search frontend. Using the load-test, we also observed that on multicore multisocket machines, there is often a tradeoff between optimizing NUMA performance by clustering threads close to the memory nodes to increase the amount of local accesses and optimizing for cache performance by spreading threads to reduce the cache contention. For example, `bigtable` benefits from cache sharing and would prefer 100 % remote accesses to 50% remote. `Search-frontend` prefers spreading the threads to multiple caches to reduce cache contention and thus also prefers 100 % remote accesses to 50% remote. **In conclusion, surprisingly, some running scenarios with more remote memory accesses may outperform scenarios with more local accesses due to an increased amount of cache contention for the latter**, especially when 100% local accesses cannot be guaranteed. **This tradeoff between NUMA and cache sharing/contention varies for different applications and when the application's corunner changes.** The tradeoff also depends on the remote access penalty and the impact of cache contention on a given machine platform. On our Intel Westmere, more often, NUMA has a more significant impact than cache contention. This may be due to the fact that this platform has a fairly large shared cache while the remote access latency is as large as 1.73x of local latency.

Previous work demonstrates the impact of cache and memory bandwidth contention for large scale WSC applications on machines with uniform memory accesses (UMA) and proposes contention-aware adaptive thread mapping [33]. In this work, we show that remote memory accesses have a significant performance impact on NUMA machines for these applications. And different from UMA machines, remote access latency is often a more dominating impact than cache contention on NUMA machines. This indicates that a simple NUMA-aware scheduling can already yield sizable benefits in production for those platforms. Based on our findings, NUMA-aware thread mapping is implemented and in the deployment process in our production WSCs. Considering both contention and NUMA may provide further performance benefit. However the optimal mapping is highly dependent on the applications and their co-runners. This indicates additional benefit for adaptive thread mapping at the cost of added implementation complexity.

[Methodology Discussion] In this work, we emphasize the importance of establishing an investigative methodology that incorporates both in production analysis and controlled load-test experimentation when investigating and quantifying the interaction between microarchitectural features and large-scale datacenter applications. Only using one of these approaches is often not sufficient for drawing meaningful conclusions.

Live production analysis avoids the challenges of setting up representative loads and replicating all aspects of the real production environment in a small-scale test environment. However, performance analysis in production WSCs is challenging as it requires automatic and lightweight monitoring of large scale systems and careful correlation and analysis of noisy data. On the other hand, while load-tests allows us to conduct fine grain and controlled studies as we can deliberately vary one or multiple performance factors while keeping other factors identical to observe the performance impact, designing a load-test that replicates production behavior is difficult and sometimes intractable. This work advocates a coarse production study as a first step to

identify evidence of a performance opportunity, followed by finer load-test studies to isolate and pinpoint the important factors related to the opportunity.

5 Related Work

A tremendous amount of research effort has been devoted to investigating the impact of NUMA on shared memory multiprocessors for various applications [4, 18, 27, 28] and designing approaches to optimizing data placement and OS scheduling for NUMA machines [1, 5, 8, 9, 19, 20, 31]. However, to the best of our knowledge, the impact of NUMA on modern large-scale datacenter applications has not been investigated and thus remains unclear. Also, most of the studies are conducted in a benchmark environment instead of a large-scale production environment. In addition, the above work does not focus on multicore architectures or take cache contention into account. As multicore, multichip servers are becoming widely used, especially as the number of processor packages increases, it is becoming necessary to revisit the impact of NUMA on the modern CMPs for some emerging workloads. Some recent work has measured NUMA-related performance in the state-of-the-art multicores using carefully designed synthetic benchmarks [11, 26]. On the other hand, there is a wealth of research related to alleviating contention in memory subsystems including cache and bandwidth on current multicores [7, 10, 15, 25, 29, 32–35]. Very recently, researchers start to investigate the tradeoffs between improving NUMA performance and reducing cache contention [3, 22]. In addition, related to constructing, scheduling and optimizing datacenters and WSCs, several prior work presents the challenges and proposed solutions [2,6,12–14,16,21,23,24,30].

6 Conclusion

This paper illustrates the opportunities, challenges and methodologies in identifying and architecting for performance opportunities in modern WSCs, especially as it relates to scheduling and thread management, through an investigation case study of how several important datacenter applications are affected by NUMA. Leveraging our newly-desgined CPU-memory locality metric and the large-scale profiling system in production, we conduct correlation analysis to quantify the NUMA impact on several web-services including Gmail backend servers in production WSCs. Our results show that NUMA locality is critical to the performance of datacenter applications and a simple NUMA-aware scheduling can yield sizable benefits. In addition to this finding in our production study, our load-test further reveals the interesting performance tradeoffs between optimizing for NUMA and reducing cache contention.

References

[1] J. Appavoo, M. Auslander, M. Butrico, D. M. da Silva, O. Krieger, M. F. Mergen, M. Ostrowski, B. Rosenburg, R. W. Wisniewski, and J. Xenidis. Experience with k42, an open-source, linux-compatible, scalable operating-system kernel. *IBM Syst. J.*, 44, January 2005.

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[3] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova. A case for numa-aware contention management on multicore systems. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, Berkeley, CA, USA, 2011. USENIX Association.

[4] T. Brecht. On the importance of parallel application placement in numa multiprocessors. In *USENIX Systems on USENIX Experiences with Distributed and Multiprocessor Systems - Volume 4*, 1993.

[5] J. Corbalan, X. Martorell, and J. Labarta. Evaluation of the memory page migration influence in the system performance: the case of the sgi o2000. ICS '03, 2003.

[6] C. Delimitrou and C. Kozyrakis. The Netflix Challenge: Datacenter Edition. Los Alamitos, CA, USA, July 2012. IEEE Computer Society.

[7] E. Ebrahimi, C. Lee, O. Mutlu, and Y. Patt. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. *ASPLOS '10*, Mar 2010.

[8] E. Focht. Node affine numa scheduler. http://home.arcor.de/efocht/sched/.

[9] B. Gamsa, O. Krieger, J. Appavoo, and M. Stumm. Tornado: maximizing locality and concurrency in a shared memory multiprocessor operating system. OSDI '99, Berkeley, CA, USA, 1999.

[10] F. Guo, Y. Solihin, L. Zhao, and R. Iyer. *MICRO 40*, Dec 2007.

[11] D. Hackenberg, D. Molka, and W. E. Nagel. Comparing cache architectures and coherency protocols on x86-64 multicore smp systems. MICRO 42, 2009.

[12] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[13] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 261–276, 2009.

[14] V. Janapa Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web search using mobile cores: quantifying and mitigating the price of efficiency. ISCA '10, 2010.

[15] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn. Using os observations to improve performance in multicore systems. *Micro, IEEE DOI - 10.1109/MM.2008.48*, 28(3):54–66, 2008.

[16] C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid. Server engineering insights for large-scale online services. *Micro, IEEE DOI - 10.1109/MM.2008.48*, 30(4):8–19, 2010.

[17] C. Lameter. Local and remote memory: memory in a linux/numa system. Linux Symposium,, 2006.

[18] R. P. LaRowe, Jr., C. S. Ellis, and M. A. Holliday. Evaluation of numa memory management through modeling and measurements. *IEEE Trans. Parallel Distrib. Syst.*, 3:686–701, November 1992.

[19] J. Laudon and D. Lenoski. The sgi origin: a ccnuma highly scalable server. ISCA '97. ACM, 1997.

[20] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn. Efficient operating system scheduling for performance-asymmetric multi-core architectures. SC '07, 2007.

[21] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. *ISCA '08: Proceedings of the 35th Annual International Symposium on Computer Architecture*, Jun 2008.

[22] Z. Majo and T. R. Gross. Memory management in numa multicore systems: trapped between cache contention and interconnect overhead. ISMM '11, New York, NY, USA, 2011. ACM.

[23] J. Mars, L. Tang, and R. Hundt. Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity. *IEEE Computer Architecture Letters*, 2011.

[24] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *MICRO '11: Proceedings of The 44th Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA, 2011. ACM.

[25] J. Mars, N. Vachharajani, R. Hundt, and M. Soffa. Contention aware execution: online contention detection and response. *CGO '10*, Apr 2010.

[26] D. Molka, D. Hackenberg, R. Schone, and M. S. Muller. Memory performance and cache coherency effects on an intel nehalem multiprocessor system. PACT '09, 2009.

[27] D. S. Nikolopoulos, T. S. Papatheodorou, C. D. Polychronopoulos, J. Labarta, and E. Ayguade;eacute;. Is data distribution necessary in openmp? Supercomputing '00, 2000.

[28] V. G. J. Peris, M. S. Squillante, and V. K. Naik. Analysis of the impact of memory in distributed parallel processing systems. *SIGMETRICS Perform. Eval. Rev.*, 22, May 1994.

[29] M. Qureshi and Y. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. *MICRO 39*, Dec 2006.

[30] P. Ranganathan. From microprocessors to nanostores: Rethinking data-centric systems. *Computer*, 44(1):39–48, 2011.

[31] L. T. Schermerhorn. Automatic page migration for linux. http://lca2007.linux.org.au/talk/197.html.

[32] L. Tang, J. Mars, and M. L. Soffa. Compiling for niceness: Mitigating contention for qos in warehouse scale computers. In *accepted to The ACM/IEEE International Symposium on Code Generation and Optimization (CGO)*, 2012.

[33] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa. The impact of memory subsystem resource sharing on datacenter applications. ISCA '11, New York, NY, USA, 2011.

[34] Y. Xie and G. Loh. Pipp: promotion/insertion pseudo-partitioning of multi-core shared caches. *ISCA '09*, Jun 2009.

[35] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. *ASPLOS '10*, Mar 2010.