2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing

Evaluation of virtual machine performance on NUMA multicore systems

Yuxia Cheng
College of Computer Science and Technology
Zhejiang University
Hangzhou, China
Email: rainytech@zju.edu.cn

Wenzhi Chen
College of Computer Science and Technology
Zhejiang University
Hangzhou, China
Email: chenwz@zju.edu.cn

***Abstract*—An increasing number of new multicore server systems use the Non-uniform Memory Access (NUMA) architecture due to its scalable memory performance. However, the multicore NUMA systems introduce complex interplay among data locality, contention on shared resources, and inter-socket data sharing overhead, which makes the achievement of an optimal and predictable system performance difficult. Virtualization technology further complicates the scheduling problem. In this paper, we first investigate the impact of virtual machine (VM) scheduling on multicore NUMA systems. Then, we analyze different VM mapping combinations and find that the best VM scheduling strategy does not only depend on the data sharing and memory characteristics of the VM itself, it is also impacted by the dynamic behavior of co-running VMs.**

***Keywords*-Multicore; Non-Uniform Memory Access; Scheduling; Virtualization;**

I. Introduction

Multicore architectures are the fundamental platforms for today's real-world systems, including high performance computing clusters, modern data centers, and cloud computing infrastructures. Multicore systems provide the advantage of simultaneous thread execution and parallel performance. However, unpredictable application performance due to contention on shared on-chip resources remains a challenging task [3] as it severely reduces the efficiency, fairness, and QoS that the platform is capable to provide.

The emerging Non-Uniform Memory Access (NUMA) architecture in new multicore systems further complicates the problem. The NUMA system links several small and cost-effective nodes via a high speed interconnect, where each node contains both processors and memory. Data locality [6] is another performance factor that should be considered when scheduling threads to different NUMA nodes. The complex interplay among contention on shared resources, data locality, and inter-node data sharing overhead makes the determination of the optimal thread-to-core scheduling difficult.

Virtualization poses additional challenges on performance optimization of the NUMA multicore systems. The guest operating system (OS) running in a virtual machine (VM) have little knowledge about the underlying NUMA multicore topology, which prevents application and OS level NUMA optimizations working effectively. Existing virtual machine monitors (VMM), such as Xen [5] and KVM [12], are unaware of the NUMA multicore topology when scheduling VMs. This NUMA-agnostic property makes it difficult to achieve efficiency, fairness, and priorities among VMs.

In this paper, we investigate the impact of virtual machine scheduling on NUMA multicore systems. Currently there is little understanding about the interaction between virtual machines and the underlying NUMA multicore systems. The contributions of our work can be summarized as follows:

(1) We analyze the existing virtual machine scheduler, the CFS scheduler used in KVM. We find that the VM scheduler cannot work effectively in NUMA multicore systems due to its NUMA-agnostic property, and find that the CFS incurs significant performance degradation as the number of threads increases in the virtualized systems.

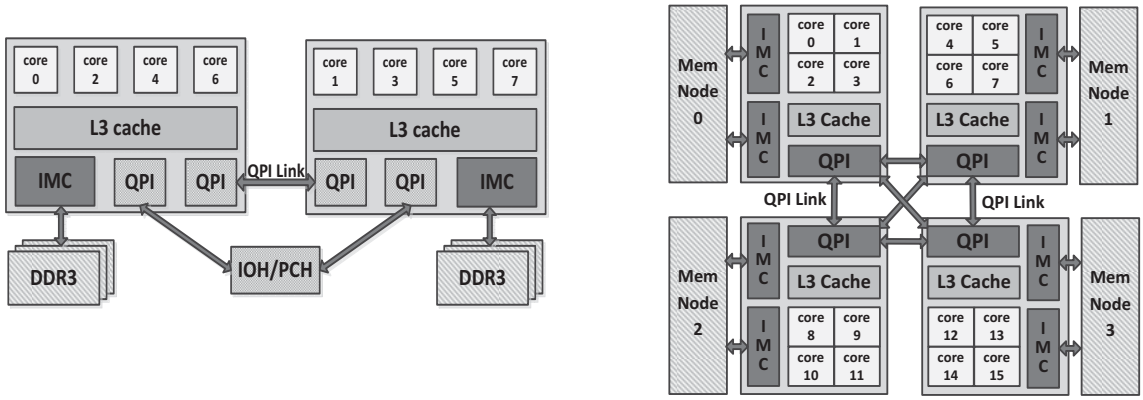
(2) We study different VM mapping combinations through our systematic evaluation. We find that the best VM scheduling strategy does not only depend on the data sharing and memory characteristics of the VM itself, it is also impacted by the dynamic behavior of co-running VMs.

The rest of this paper is structured as follows: the new NUMA multicore architecture is described in section 2; An analysis of the KVM scheduler is presented in section 3; In section 4, we evaluate the impact of VM mapping combinations and analyze the experimental results. In section 5, we describe the related work. Finally, we conclude and discuss the future work in section 6.

II. The NUMA Multicore Architecture

Multicore processors are commonly seen in today's computer architectures. However, a high frequency (typically 2-4 GHz) core often needs an enormous amount of memory bandwidth to effectively use its processing power. Even a single core running a memory-intensive application will find itself constrained by memory bandwidth. As the number of cores becomes larger, this problem becomes more severe on Symmetric Multi-Processing (SMP) systems, where many cores must compete for memory controller and bandwidth in a Uniform Memory Access (UMA) manner. The Non-Uniform Memory Access (NUMA) architecture is then proposed to alleviate the constrained memory bandwidth problem as well as to increase the overall system throughput.

978-0-7695-5094-7/13 $31.00 © 2013 IEEE
DOI 10.1109/3PGCIC.2013.27



(a) 2-NUMA-node Westmere-EP System

(b) 4-NUMA-node Westmere-EX System

Figure 1. The Non-Uniform Memory Access (NUMA) multicore System Architecture

The NUMA system links several small and cost-effective nodes via a high speed interconnect, where each node contains both processors and memory. Each node has an advanced memory controller that allows a node to access memory on all other nodes. When a processor accesses memory data that is not in its own node, the data must be transferred through the NUMA interconnect from the remote memory node. Processors accessing local node memory is faster than accessing remote node memory, thus the system has Non-Uniform Memory Access latency. Currently most commodity servers are using the NUMA architecture due to its high scalability and cost-effective properties.

The 2-NUMA-node Intel Xeon Westmere-EP topology is shown in Fig.1(a). In the Westmere-EP architecture, there are usually four or six cores sharing the Last Level Cache (LLC, or L3 cache) in a socket, while each core has its own private L1 and L2 cache. Each socket has the Integrated Memory Controller (IMC) connected to the local three channels of DDR3 memory. Accessing to the physical memory connected to a remote IMC is called the remote memory access. The Intel QuickPath Interconnect (QPI) interfaces are responsible for transferring data between two sockets. And the two sockets communicate with I/O devices in the system through IOH/PCH (IO Hub / Platform Controller Hub) chips. Fig.1(b) shows a 4-NUMA-node Intel Xeon Westmere-EX topology, there are four NUMA nodes interconnected by the QPI links in the system, and each node has four cores sharing one LLC with two IMCs integrated in the socket. Although other NUMA multicore processors (e.g., AMD Opteron) may differ in the configuration of on-chip caches and the cross-chip interconnect techniques, they have similar architectural designs.

Recent studies [6], [17], have shown performance degradations on NUMA multicore systems due to placing applications onto improper NUMA nodes. We summarize the performance degradation factors into three aspects:

(1) Remote data access. If a thread is running on a core belonging to one NUMA node and the thread's memory is located in another NUMA node, it will cause the thread accessing remote data, thus increasing memory access latency.

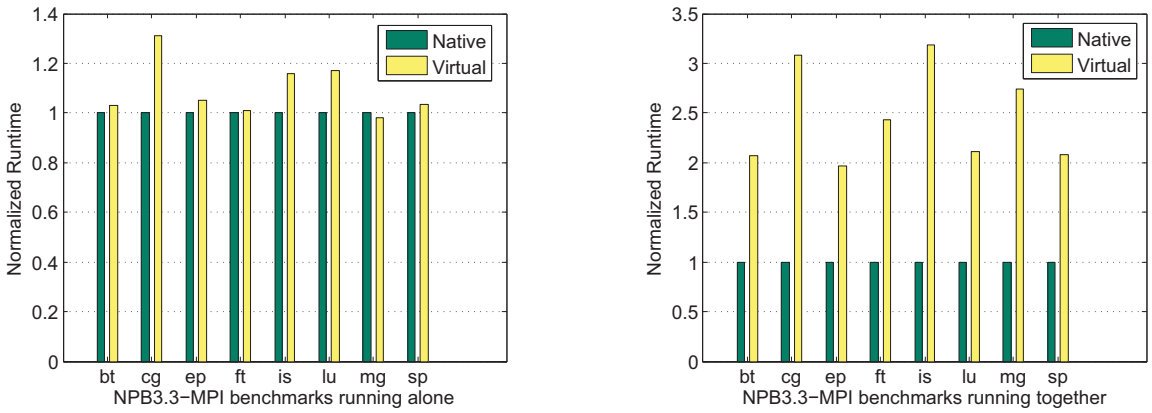
(2) Contention on shared on-chip resources. If two memory intensive threads are scheduled into two cores that shares the same LLC, memory controller, and other hardware prefetching units etc., it will cause severe contention on these shared on-chip resources, thus resulting the performance degradation of both threads.

(3) Inter-socket data sharing overhead. If two threads that frequently access shared data are scheduled into two cores that resides in different sockets (also different NUMA nodes in the Westmere architecture), it will cause inter-socket data sharing overhead. Because the shared data consistency messages need to be transferred by the QPI link between two sockets.

The complex interplay of these three performance factors on NUMA multicore systems makes the achievement of optimal and predictable system performance difficult. Therefore, it is important to study the multicore NUMA effect on modern servers, especially when these servers are deployed of virtualized systems with various VMs running on them. In the following sections, we will discuss the performance impact of virtual machine scheduling on NUMA multicore systems.

III. Analysis of Existing VM schedulers

In this section, we analyze the existing VM schedulers and evaluate the performance impact of scheduling VMs in the NUMA multicore system. We compare the native benchmark performance with the virtual performance where benchmarks are running inside the VM. We find that the performance of benchmarks running in the virtual machine degrades slightly



(a) Single-thread Case

(b) 4-thread Case

Figure 2. Performance Comparison of Scheduling Impact on Virtual vs. Native Machines

compared with running in native machines when the number of running threads is equal or less than the number of physical cores in the system. However, when the number of running threads is much greater than the number of cores in the system, the performance of benchmarks running in the virtual machine decreases significantly compared with running in the native machine. This observation indicates the virtual machine scheduling has great impact on the performance of virtualized NUMA multicore systems.

To evaluate the effects of virtual machine scheduling, we use the KVM (Kernel-based Virtual Machine) as the virtualization environment. In section 3.1, we analyze the Linux CFS (Completely Fair Scheduler) that used in the KVM monitor as the VCPU scheduler. In section 3.2, we compare the scheduling impact on the virtual performance and native performance in NUMA multicore system, and present our experimental results.

A. The CFS Scheduler

The scheduling module of the KVM virtual machine monitor depends on the CFS scheduler of the Linux kernel. The CFS scheduler treats a VM instance as a normal process in the Linux system, and treats each VCPU in the VM as a thread of the VM process. It is the CFS's responsibility to schedule virtual machines (more precisely VCPUs) in the system. CFS is a virtual runtime based multicore task scheduler with the distributed run-queue structures. A thread's virtual runtime is defined as the thread's cumulative runtime inversely scaled by its weight. Each core maintains a run-queue of runnable threads, and a CFS instance runs periodically to make scheduling decisions according to the virtual runtime of threads in each core. CFS attempts to give each thread CPU time proportional to its weight, and uses a weight-based load balancing mechanism to fairly distribute system load among run-queues. Threads will migrate among different cores, which may incur extra overhead when thread migration occurs frequently in the system, especially in a NUMA multicore system.

Current CFS scheduler is unaware of the underlying NUMA multicore topology. Thus, NUMA-agnostic thread scheduling, particularly when this thread is a VCPU thread of a virtual machine, causes ineffective use of shared on-chip resources, remote data access, and inter-socket data sharing overhead. In the next section, we present an experimental analysis of scheduling impact on virtual machine performance in the NUMA multicore system.

B. Scheduling Impact on Virtual Performance

We conduct an experiment to study the scheduling impact on virtual machine performance in the NUMA multicore system. The experimental results were gathered on Dell R710 server with two 2.40 GHz Intel (R) Xeon (R) CPU E5620 processors based on the Westmere-EP architecture. Each E5620 processor has four cores sharing a 12MB L3 cache. The R710 server has a total of 8 physical cores and 16GB memory, with each NUMA node having 4 physical cores and 8 GB memory. The virtual machine used in this experiment is configured with 8 VCPUs and 16GB memory that is the same as the physical machine. We use the NAS Parallel Benchmarks (NPB) [1] to run on the R710 server. The host and guest operating systems are the SUSE 11 SP2 (the Linux kernel 3.0.24).

First, we simultaneously run 8 **single-threaded** NPB-MPI benchmarks in the native Linux machine. We run the 8 benchmarks together for 5 times and calculate the average runtime of each single-threaded benchmark. Then, we put the same 8 benchmarks into the 8-VCPU virtual machine, and the guest OS in the VM is the same with the host OS. We also run the 8 benchmarks simultaneously for 5 times and calculate the average runtime. The experimental result is shown in Fig.2(a), we use each native benchmark's runtime as the base time and normalize the virtual benchmark's

runtime using the base time. We find that 5 benchmarks (*bt, ep, ft, mg, sp*) running in the VM have almost the same performance as running in the native machine. And 3 benchmarks (*cg, is, lu*) running in the VM have some performance degradation compared with that running in the native machine, the degradation ratios are 31%, 16%, and 17% respectively.

Second, we simultaneously run 8 **four-threaded** NPB-MPI benchmarks in the native Linux machine and calculate their average runtime. Then, we run the same 8 benchmarks simultaneously in the VM and also calculate their average runtime. Therefore, both in the native machine and in the virtual machine, the number of running threads is 4 times larger than the number of available physical CPUs (PCPUs) and virtual CPUs (VCPUs) respectively. In this case, both the native and virtual system must schedule the running threads frequently to let all threads have corresponding CPU time slices and to balance CPU load. The experimental result is shown in Fig.2(b). This time, we find that all the benchmarks running in the VM have significantly longer runtime than in the native case. An average of 2.5 times performance degradation in the virtual machine than in the native machine. Particularly, the runtime of *cg* and *is* running in the VM have 3.08 and 3.18 times longer than in the native machine respectively.

From the experiment, we observe that the scheduling overhead in the virtual machine is significantly higher than in the native machine. If the scheduling operation is less frequent (running single-threaded benchmarks as shown in Fig.2(a)), the virtual performance is close to the native performance. However, when the number of running threads in the system becomes larger, the benchmark performance in the VM decreases significantly. This is because that the Virtual Machine Monitor (VMM, the KVM monitor in this experiment) and VM is unaware of the underlying NUMA multicore topology. This NUMA-agnostic property will cause the VMM scheduling VCPU threads to inappropriate PCPUs and the VM scheduling application threads to inappropriate VCPUS, which incurs data locality, contention for shared on-chip resources, and more inter-socket data sharing overhead. Whats more, frequent scheduling operations incur much more context switch (extra VM exit and VM entry context switch) overhead than in the native machine. In the next section, we will evaluate the performance impact factors of virtual machine scheduling in the NUMA multicore systems.

IV. Performance Evaluation

In this section, we evaluate the memory characteristics of virtual machines that running different NPB benchmarks and study different VM mapping combinations that impact the system performance. Through our systematic evaluation , we find that the best VM scheduling strategy does not only depend on the data sharing and memory characteristics of the VM itself, it is also impacted by the dynamic behavior of co-running VMs.

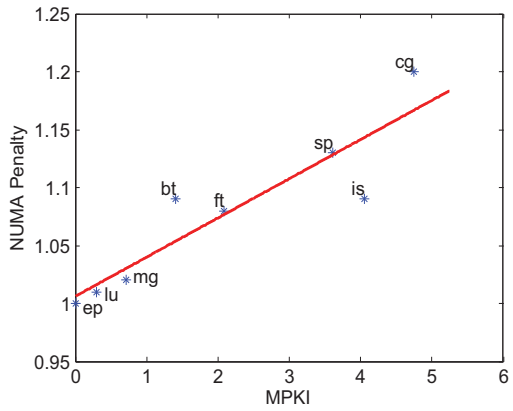
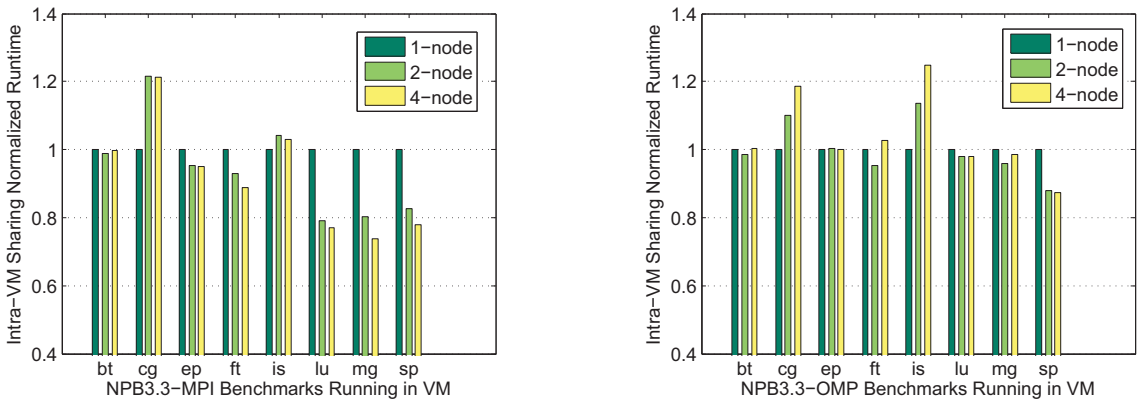


Figure 3. The Memory Characteristics of NPB benchmarks

A. Experimental Platforms

We use two experimental systems for evaluation. One is the two-NUMA-node Dell R710 server used in section 2 and section 3. The other is a four-NUMA-node Dell R910 server with four 1.87 GHz Intel (R) Xeon (R) CPU E7520 processors based on the Nehalem-EX architecture. Each E7520 processor has four cores sharing a 18MB L3 cache. The total memory is 64 GB with each NUMA node having 16 GB memory. Both the host and guest operating system used in the experiment are SUSE 11 SP2 (the Linux kernel version 3.0.13). KVM module is used in the kernel as the virtualization environment. We use NAS Parallel Benchmark (NPB3.3) to test virtual machine performance. NPB benchmark suite is a set of benchmarks developed for the performance evaluation of highly parallel supercomputers. These benchmarks consist of five parallel kernels and three simulated application benchmarks.

We use the perf [18] tool to measure the memory characteristics of the NPB benchmarks. We use NUMA penalty and MPKI (L3 cache misses per kilo instructions [6]) to indicate a benchmark's memory intensiveness. The NUMA penalty of a program is the slowdown of a remote execution relative to the program's local execution. The NUMA penalty is calculated as CPIremote/CPIlocal, where CPIremote denotes the CPI (cycles per instruction) of a program executing remotely, CPIlocal denotes the CPI of a program executing locally. We use the perf tool to measure the benchmark's CPI online. The MPKI of a program is used to estimate cache pressure of the program. We also use the perf tool to measure the benchmarks MPKI online. As shown in Fig.3, we plot the NUMA penalty of eight NPB benchmarks against their MPKI, and also plot the linear model fitted onto the data. The results show that the two parameters are positively correlated, the NUMA penalty increases with the



(a) Intra-VM MPI

(b) Intra-VM OMP

Figure 4. Performance Variation of Intra-VM VCPU Mapping Strategies

MPKI. We can divide the eight NPB benchmarks into three classes according to their memory intensity (the value of their MPKI): Class A includes cg, is, and sp benchmarks, which has high memory intensity (MPKI greater than 3); Class B includes bt and ft, which has medium memory intensity (MPKI between 1 and 3); Class C includes ep, lu, and mg, which has relatively low memory intensity (MPKI less than 1).

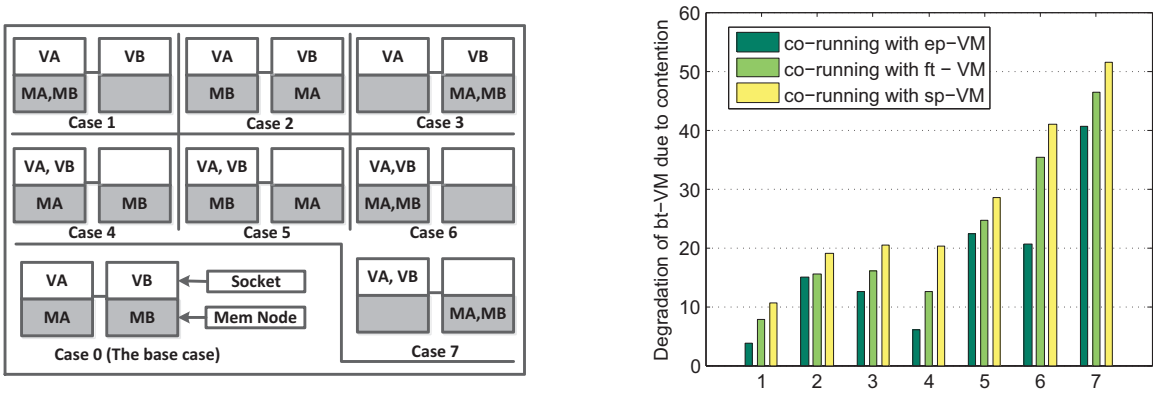
B. Intra-VM VCPU Mapping

Virtual machines are now typically configured with multiple VCPUs. And the VCPUs inside a VM will be scheduled into different NUMA nodes according to the online system load. In order to analyze the performance impact of VCPU scheduling within a VM, we conduct an experiment to investigate the different VCPU mapping strategies that may cause different performance variations due to data locality, contention on shared resources, and inter-socket sharing overhead.We use the four-NUMA-node R910 server as the experimental platform, and the virtual machine is configured with 4-VCPU and 16 GB memory.

In the **first** case, we bind all the 4 VCPUs on one NUMA node and bind the memory of the VM on the same node. We run the eight NPB-MPI and NPB-OMP benchmarks respectively. Each time we run one of the eight 4-threaded NPB benchmarks inside the VM and record their average runtime (each benchmark running three times). In the **second** case, we bind 2 VCPUs of the VM on NUMA node-0 and the other 2 VCPUs of the VM on NUMA node-1. The VM's memory allocation strategy uses the default Linux NUMA memory management that allocates a threads memory in the local NUMA node where the thread is running. We also run the MPI and OMP benchmarks and record their average runtime. In the **third** case, we bind 4 VCPUs on 4 NUMA nodes respectively, that is each NUMA node has one VCPU running on them. The memory allocation strategy is the same as in the second case. We also run the MPI and OMP benchmarks and record their average runtime. The experimental result is shown in Fig.4. We plot the normalized runtime of each benchmark using the 1-node case (4 VCPUs all bind on one NUMA node) as the base case. The 2-node and 4-node case represent the second and the third case described above.

We find that different intra-VM VCPU mapping strategies have significant performance impact on the virtualized NUMA multicore system. As shown in Fig.4(a), the *mg* benchmark's runtime in 4-node case is 27% shorter than in 1-node case, and other benchmarks (*cg, ft, lu, sp*) also show significant performance variation from 12% to 23%, the rest of benchmarks (*bt, ep, is*) show minor performance variation from 2% to 5%. In Fig.4(b), the performance variation is also very significant. The benchmark *cg*, *is*, and *sp* have large performance variation from 13% to 25%. And benchmark *bt*, *ep*, *ft*, *lu*, and *mg* also have performance variation from 2% to 5%.

Benchmarks *cg* and *is* both in the MPI and OMP tests show shorter runtime in 1-node case than in 2-node and 4-node cases. This is because *cg* and *is* have larger MPKI (see Fig.3) that will cause frequent main memory access (serve cache misses from LLC), therefore both *cg* and *is* favor data locality (accessing memory from local NUMA node) to reduce main memory access latency. The runtime variation of *is* in the OMP test is larger than in the MPI test, because MPI and OMP have different parallel programming model, MPI is the message-passing based model while OMP is the shared-memory based model. Therefore, the OMP benchmark test favors more data locality factor. However, the benchmark *lu*, *mg*, and *sp* show longer runtime in 1-node case than in 2-node and 4-node cases. This is because *lu*, *mg*, and *sp* have large working set, and they favor large cache size more than data locality factor to improve their performance. Therefore, when distributing the 4 VCPUs into



(a) 8 Inter-VM Binding Cases

(b) Inter-VM Binding Cases

Figure 5. Performance Impact of Inter-VM Resource Contention

2 or 4 separate NUMA nodes (each node has separate large 18 MB L3 cache size), the *lu*, *mg*, and *sp* shows performance improvement compared with putting all VCPUs into one NUMA node.

In Fig.4(b), the benchmark *ft* and *mg* clearly show the complex interplay of three performance factors: data locality, contention on shared L3 cache, and inter-socket data sharing overhead. Both *ft* and *mg* have shorter runtime in 2-node case than in 1-node and 4-node cases. This shows the trade-off among the three performance factors. The 1-node case has strong data locality, little inter-socket data sharing overhead, but has strong contention on shared on-chip resources. However, the 4-node case has little contention on shared on-chip resources, but has weak data locality, and large inter-socket data sharing overhead. To a certain degree, the 2-node case is a balance among the three performance factors.

From the experiment, we find that when the performance factors have different contributions to the overall system performance, it is the dominant factor that typically determines the best mapping strategy. However, when the performance factors have equal contributions to the overall system performance, the best mapping strategy is usually a trade-off among all the performance factors. This finding helps the design of future NUMA-aware VCPU scheduling algorithms to improve virtual machine performance in NUMA multicore sytems.

V. Inter-VM Resource Contention

In this section, we analyze the performance impact of inter-VM resource contention. Multiple virtual machines are usually consolidated onto a physical NUMA multicore server. These VMs will contend for underlying shared physical resources. The performance degradation caused by inter-VM resource contention can be very significant and will be changed due to different program behavior of various VMs.

We conduct an experiment to show the performance impact of inter-VM shared resource contention. We use two VMs to concurrently run on the multicore NUMA system and compare the performance variation due to different VM mapping strategies and different benchmarks running inside VMs. The two VMs (VM-A and VM-B) have the same configuration with 2 VCPUs and 8 GB memory. The experimental results were gathered on the 2-NUMA-node R710 server. We use *numactl* command to bind VM's VCPU threads to sockets and their memory to NUMA nodes, then we enumerate eight VCPU-to-socket and memory-to-node binding cases, and Fig.5(a) shows the eight different binding cases. Case 0 is the base case that VM-A's VCPUs (short for VA) are bound to socket-0 and its memory (short for MA) is bound to node-0, VM-B's (short for VB) VCPUs are bound to socket-1 and its memory (MB) is bound to node-1. The rest of the seven cases are shown in Fig.5(a). We use the VM-A as the target VM that the bt benchmark is running inside it, and use the VM-B as the co-running VM that each time one of three selected NPB benchmarks is running inside it. The three benchmarks are *ep*, *ft*, and *sp*, and they are selected according to the classification discussed in section 4.1, representing low, medium, and high memory intensity of co-running VMs respectively.

The experimental results are shown in Fig.5(b). We plot the performance degradation of the *bt* benchmark inside VM-A in Case 1-7 relative to Case 0 when co-running with VM-B. The results show that the performance degradation can be as high as 51.4% in Case 7 that sp is the benchmark co-running inside VM-B. Other cases also show significant performance degradation of the *bt* benchmark due to different resource contention, cases 1, 3, 6, and 7 have memory controller contention; cases 4, 5, 6, and 7 have LLC contention; cases 2, and 7 have interconnect contention; cases 2, 3, 5, 7 have remote access latency. Fig.5(b) shows the performance variations due to these inter-VM shared

resources contentions. From the result, we observe that the performance degradation caused by inter-VM resource contention can be very significant and different resource contention cases show different performance reductions. What's more, the performance of the target VM is also impacted by the dynamic behavior of co-running VMs.

VI. Related Work

Much research has focused on NUMA-related system performance in OS and architecture research field [3], [6], [8], [13], [17], etc. Some research efforts [8] and [13] try to address effective co-location of the computation and the related memory on the same node. Some studies [3], [6], [17] have shown significant performance impact of shared resource contention in the context of multicore NUMA systems. However, to our best knowledge, very few efforts have been made to efficiently address the challenges of NUMA multicore performance variation in virtualized systems. Our work studies the impact of virtual machine scheduling on new NUMA multicore systems and provide meaningful observations for the future design of NUMA-aware virtual machine scheduling algorithms in the VMMs.

The performance evaluations of virtualized systems have been studied in recent years. There has been a lot of research in assessing the performance of virtualized systems in cloud computing environments [7], [11]. Compared to this body of previous work, ours is different in granularity. The granularity is VCPU/PCPU and the underlying shared physical resources in our work compared to the black box VM rent from cloud service provider. We conduct extensive experiments on inter-VM VCPU mapping and intra-VM resource contention effect on the multicore NUMA systems using NPB benchmarks.

The HPC community has shown much interest for the use of virtualization recently. Studies of [15] and [10] find that virtualization overhead is negligible for compute intensive HPC kernels and NPB benchmarks; In [10], W. Huang et al. described a case for high performance computing with virtual machines and showed that I/O virtualization overhead is the major factor for performance degradation of NPB benchmarks. Other studies [16] and [14] discuss challenges and issues in system virtualization technologies with emphasis on their value in HPC environments. Researchers in [4] and [9] evaluate the effect of HPC applications in virtualized VMware ESXi and Xen systems, and they present simple performance variation results due to different VM configurations. In contrast to these, our work evaluates the performance impact of virtual machine scheduling in the KVM system on new NUMA multicore servers, and we reveal three performance degradation factors and two kinds of VM running scenarios (inter-VM and intra-VM) that essentially impact on the system performance.

VII. Conclusion

We evaluated the performance impact of virtual machine scheduling on the NUMA multicore systems. We found that the CFS scheduler used in KVM cannot work effectively in NUMA multicore systems due to its NUMA-agnostic property. And we presented a detailed performance evaluation of virtual machines running on the NUMA systems. We discovered that the best VM scheduling strategy does not only depend on the data sharing and memory characteristics of the VM itself, but also impacted by the dynamic shared physical resource contention of co-running VMs. These observations will motivate the future design of new virtual machine schedulers.

References

[1] The NAS Parallel Benchmarks, http://www.nas.nasa.gov/publications/npb.html

[2] J. Ahn, C. Kim, J. Han, Y. Choi, and J. Huh, "Dynamic virtual machine scheduling in clouds for architectural shared resources," in *HotCloud*, 2012.

[3] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 1, pp. 129–142, 2010.

[4] Q. Ali, V. Kiriansky, J. Simons, and P. Zaroo, "Performance evaluation of hpc benchmarks on vmware's esxi server," in *Euro-Par*, 2012.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 164–177, 2003.

[6] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova, "A case for numa-aware contention management on multicore systems," in *USENIX ATC*, 2011.

[7] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *ICS*, 2008.

[8] M. Ghosh, R. Nathuji, M. Lee, K. Schwan, and H. Lee, "Symbiotic scheduling for shared caches in multi-core systems using memory footprint signature," in *ICPP*, 2011.

[9] J. Han, J. Ahn, C. Kim, Y. Kwon, Y. Choi, and J. Huh, "The effect of multi-core on HPC applications in virtualized systems," in *Euro-Par* , 2010.

[10] W. Huang, J. Liu, B. Abali, and D. Panda, "A case for high performance computing with virtual machines," in *ICS*, 2006.

[11] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 931–945, 2011.

[12] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: the Linux virtual machine monitor," in *Proceedings of the Linux Symposium*, 2007.

[13] Z. Majo and T. Gross, "Memory management in numa multicore systems: Trapped between cache contention and interconnect overhead," in *ACM SIGPLAN Notices*, vol. 46, no. 11, pp. 11–20, 2011.

[14] M. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for high-performance computing," in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 2, pp. 8–11, 2006.

[15] J. Moses, R. Iyer, R. Illikkal, S. Srinivasan, and K. Aisopos, "Shared resource monitoring and throughput optimization in cloud-computing datacenters," in *IPDPS*, 2011.

[16] J. Simons and J. Buell, "Virtualizing high performance computing," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 4, pp. 136–145, 2010.

[17] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *ISCA*, 2011.

[18] S.Eranian, "What can performance counters do for memory subsystem analysis?" in MSPC, 2008.