Virtual Machine Scheduling for Architectural Shared Resources in Open Stack Based Cloud

Jayalakshmi N Sagar B M

Abstract*— In IAAS cloud virtual machine (VM) migration between socket and nodes has been used to avoid conflicts among VM on system resources such as CPU and memory. Micro-architectural resources such as caches, memory controllers and non-uniform memory access (NUMA) affinity, have relied on intra socket scheduling to reduce cache contention. This paper proposes algorithm for cluster-level virtual machine scheduling based on cache sharing on same socket and these are considered in dynamic environment which do not require any prior knowledge on nature of VMs.The paper would present algorithm for the scheduler into a real cloud system (Open Stack), and the performance of the scheduler will then be investigated on various scientific and commercial workloads.

Keywords*—architecture, NUMA, OpenStack , CPU bound, memory bound.

I. INTRODUCTION

This paper report on scheduling virtual machines based on performance and work load analysis for an Open Stack based IAAS architecture. In any cloud systems based on virtualization, virtual machines (VM) share physical resources. Although physical resource sharing can improve the overall utilization of existing resources, contentions on physical existing resources often lead to significant performance degradation. To reduce the effect of such contentions, cloud systems employs dynamic rescheduling of VMs with live migration techniques [3], changing the position of running VMs between cores and socket. Such VM migration has been used to resolve conflicts or balance load on system resources such as CPUs, memory, and I/O sub-systems. VM migration can be triggered by monitoring the usages of these resources for VMs in a cloud system [4, 8]. The multi-cores architecture has enabled the sharing of micro-architectural resources such as shared Last Level caches and memory controllers. Contention on micro-architectural resources has been considered as a major reason for performance variation, as an application can be affected by other applications on same cache even though it receives the same share of CPU, memory, and I/O. For a single system, there have been several prior studies to reduce the impact of contention on shared Last level caches and memory by carefully scheduling threads [2, 9]. The technique is to group applications to share a cache to mitigate the overall cache misses for a system. A cloud systems with virtualization, provides an opportunity to enhance the scope of contention-aware scheduling, as virtual machines can be scheduled with live migration.

Manuscript received 0n May 2013.

Jayalakshmi N, MTECH [SE], 4th Semester, Department of Information Science and Engineering, R V College Of Engineering, India,Bangalore–560059,India.

Sagar B M, Associate Professor Department of Information Science and Engineering, RV College Of Engineering ,India, Bangalore–560059,India.

This paper uses live VM migration and dynamically schedule VMs for minimizing the contention on shared caches and memory.

This paper proposes contention-aware VMs scheduling technique for cache sharing. This techniques consider the cache behaviour of VMs at runtime, and dynamically migrate VMs between cores and sockets, if the current position of VMs is causing shared cache conflicts or wrong NUMA affinity. Since the techniques identify the VM behaviour on-line and resolve conflicts with live migration from socket or node, they do not require any prior knowledge on the nature of VMs that has to be scheduled. The **cache aware cloud scheduling** algorithm reduces the last level cache (LLC) misses in a particular socket of cloud system. This concept can be evaluated using selected SPECcpu 2006 applications in various combinations.

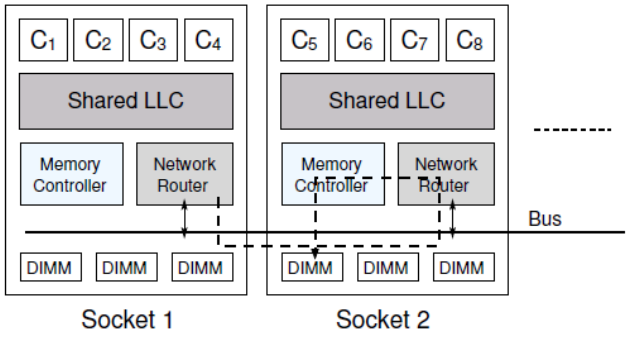

Figure 1: Shared caches and NUMA

II. MOTIVATION

2.1 Cache Sharing

Although shared caches among cores can potentially increase the efficiency with dynamic capacity sharing among all cores, they also introduce contention when one of cores produces excessive cache misses and prevents the use of cached data from the other cores. Figure 1 shows a multi-core server system with multiple sockets. In each socket, there will be a shared last-level cache (LLC), and all cores accesses memory across different sockets which have longer latencies than those to the local socket. There have been discussed several studies to reduce such interferences in shared caches with partitioning Last Level cache [6, 7] or scheduling threads [5, 9]. In the all scheduling solutions [9], threads are made into groups and mapped to different sockets, aiming to reduce the sum of cache misses of all the shared LLCs. In the scheduling policy in this paper for a system with two sockets, threads are sorted based on their LLC misses, and grouped into set of two with similar or same sum of LLC misses.

Minimizing the difference between LLC misses among the last-level caches mitigate the overall LLC misses in the system. However, NUMA affinity is more complicated for such cache-aware scheduling. If an application is running on different socket from the socket in which its memory pages reside, the cost of LLC misses will increase due to the NUMA effect. Therefore, scheduling to minimize the overall cache misses must also consider possible NUMA effects In virtualized systems, commercial hypervisor provides dynamic page migration to reduce memory access latencies for VMs, but it does not consider cache sharing effect [1].The prior studies use thread scheduling in a single system to reduce the shared cache contention and negative NUMA effects. Such intra-core and intra socket scheduling limits the opportunity to know the best group of threads sharing an LLC within a system. However, in a virtualized cloud system composed of a large number of nodes, VMs can migrate across physical system (or node) boundaries, potentially increasing the chance to find a better grouping of VMs for shared Last Level Caches, and support NUMA affinity.

2.2 Performance Implication in Clouds

This section quantitatively intended to show the performance implication of cache sharing and NUMA affinity in a small scale cloud system. This implication uses a 4-node cluster with 8 cores in two sockets for each node. The details of the experiments are shown in Section 4.1. Table 1 presents the performance of a combination of workload from 4 application types on the cluster, with 8 VM instances for each combination of application type. For the cache sharing case, the best case is to map VMs to cores so that the sum of LLC misses of all the sockets in the cloud system is reduced. The worst case is the mapping VMs with the highest difference between the largest and smallest per-socket LLC misses in the cloud system. milc and GemsFDTD could increase their performance by sharing caches with other less memory-intensive workloads. Similarly, hmmer and namd have no improvements in performance, since these benchmarks do not require high capacity for LLCs.

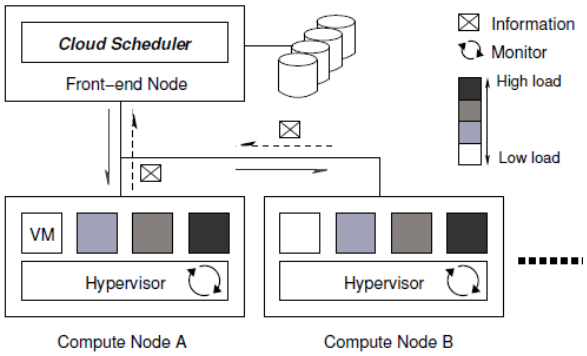

Figure 3: Memory-aware Cloud Scheduler

In a large scale cloud system, the heterogeneity of VM cache behavior's across different nodes, is expected to increase, as various customers will share the cloud. Exploiting such heterogeneity of cache behavior's and memory-aware cloud scheduling can potentially improve the efficiency of shared cache and NUMA affinity, avoiding the worst case scheduling.

III. MEMORY-AWARE CLOUD SCHEDULING

For memory-aware scheduling, the cloud scheduler collects the cache behavior of each VM from computing nodes, and migrates VMs if such migration can potentially reduce the overall cache misses and the average memory access latencies by NUMA affinity in the cloud system. Figure 3 explains the overall architecture of the memory-aware cloud scheduler. In each computing node, a monitor checks LLC misses with hardware performance monitoring counters, and periodically sends the per-VM LLC miss and NUMA affinity information to the cloud scheduler. Based on the VM status information from all the nodes, the cloud scheduler makes global scheduling decisions.

This paper presents scheduling policy for **cache-aware scheduler** which considers only the contentions on shared caches, ignoring the NUMA effect. This policy will group VMs to minimize the overall LLC misses in the entire cloud system, even if the grouping can violate NUMA affinity. One of the advantages of the proposed memory-aware scheduler is that they use only the information of VMs measured dynamically, without previous knowledge on behavior of the VMs. The memory-aware cloud schedulers initially place VMs on computing nodes, only considering CPU and memory availability for each node. However, they dynamically identify the cache behavior of the VMs, and re-locate them to improve the memory behavior.

Algorithm 1: Pseudo code for Cache-aware scheduler
PList =< *pm*1, ..., *pmn* > // LLC misses of all compute nodes
VList =< *vm*1, ..., *vmk* > // LLC misses of VMs in a node
/* Step1: local phase */
for each node *i* in 1 ... *n* **do**
// gather Last Level Cache misses for all VMs in node *i*
pmi (**gather (** *i* **)**
VList (**sort (** *pmi* **)**
// distribute the VMs such that sockets with even LLC misses
distribute (*VList*)
end for
/* Step2: global phase */
// find nodes in entire cloud with the largest and smallest LLC misses
maxNode(**findMaxNode** (*PList*)
minNode(**findMinNode** (*PList*)
// find VMs with largest and smallest Last Level Cache misses from two nodes
maxVM (**findMaxVM** (*maxNode*)
minVM (**findMinVM** (*minNode*)
if *maxNodeLLC* −*minNodeLLC* > *threshold* **then**
swap (*maxVM*, *minVM*)
end if

Cache-Aware Scheduler: The cache-aware scheduler relocates VMs to minimize the overall Last Level Cache misses in the cloud system. It employs both local and global scheduling phases. In local phase, VMs in each node are grouped and scheduled to shared cache (sockets) in the node. Since VM migrations across physical nodes consume network bandwidth and computational capability, we attempt to minimize such VM migration by optimizing VM scheduling within a node first. In the global phase, the cloud scheduler attempts to re-distribute VMs to have even LLC misses in all the nodes in the cloud system.

Algorithm 1 presents the cache-aware scheduling with the two phases, ie in the local phase, VMs in each node are sorted by LLC misses, and then grouped to make each LLC have even misses. We use the same simple algorithm used by Zhuravlev et al [9]. For example, for a node with two shared cache domains, the VM with the largest number of LLC misses is assigned to the first group, and the second VM is assigned to the second group. Among the remaining VMs, the VM with the smallest number of LLC misses is assigned to the first group, and the VM with the second smallest number of LLC misses is assigned to the second group. This continues until all VMs are assigned to one of the two groups. In the global phase, the scheduler finds two nodes, in the cloud system, with the largest and smallest numbers of LLC misses. From the two nodes, it finds two VMs with the largest and smallest numbers of LLC misses, respectively. If their LLC miss difference is larger than a threshold, the two VMs are swapped by live migration. The scheduler periodically executes the two-phase scheduling to gradually reduce the overall LLC misses in the cloud system.

Table 1: Selected benchmarking workloads from SPECcpu 2006

	Memory bound		**CPU-bound**	
1	GemsFDTD	milc	Hmmer	namd
2	Omnetpp	lbm	Gobmk	sjeng
	Memory bound			**CPU-bound**
3	cactusADM	gcc	Soplex	namd
	Memory bound	**CPU-bound**		
4	libquantum	tonto	Povray	sjeng
Memory bound				
5	cactusADM	milc	Omnetpp	soplex
CPU bound				
6	gobmk	sjeng	Namd	povray

IV. EVALUATION

4.1 Methodology

We have implemented the proposed schedulers running in a separate cloud manager node. Each computing node is virtualized with the open source Xen hypervisor. Each node runs a monitoring tool, which records LLC misses for VMs and periodically sends the miss and NUMA affinity information to the cloud scheduler. On top of the Xen hypervisor, each node runs 8 guest VMs, which use a Ubuntu distribution based on Linux kernel 2.6.18. In our small scale test bed, there are 4 physical nodes with total 32 VMs. Each physical node has 8 cores placed on two chips (sockets) and each socket on a node has a 12MB L3 cache shared by 4 cores. In the dual-socket system, memory access latencies to the local socket and remote sockets are different. Each VM employs a single core and 1GB guest physical memory size. Table 1 presents our benchmark applications. Analysis create 6 workloads by mixing applications with various memory characteristics. Each workload has 4 different benchmarking applications, and 32 VMs run 8 instances of each application.

4.2 Performance Improvements

Performance can be measured in terms of LLC misses and Inter process communication of VMs where these benchmarks are running. When each VM is pinned to physical core of same socket and benchmarks of various combinations are experimented. milc with lbm pinned to same socket will give more LLC miss rate, which is a special case to be considered.

V. CONCLUSIONS AND FUTURE WORK

This paper proposed memory-aware cloud scheduling technique, which do not require any prior knowledge on the behaviours of VMs. This paper shows that VM live migration can also be used to mitigate micro-architectural resource contentions, and the cloud-level VM scheduler must consider such hidden contentions.

REFERENCES

1. VMware ESX Server 2 NUMA Support. White paper. .
2. BLAGODUROV, S., ZHURAVLEV, S., MOHAMMAD, D., AND FEDOROVA,A. A case for numa-aware contention management on multicore processors. In Proceedings of the USENIX Annual Technical Conference (2011).
3. CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (2005).
4. GULATI, A., SHANMUGANATHAN, G., HOLLER, A., AND AHMAD,I. Cloud-scale resource management: challenges and techniques.In Proceedings of the 3rd USENIX conference on Hot topics in cloud computing (2011).
5. MERKEL, A., STOESS, J., AND BELLOSA, F. Resource conscious scheduling for energy efficiency on multicore processors. In Proceedings of the 5th European conference on Computer systems (2010).
6. QURESHI, M. K., AND PATT, Y. N. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In Proceedings of the 39th Annual IEEE/ACMInternational Symposium onMicroarchitecture (2006).
7. SUH, G. E., DEVADAS, S., AND RUDOLPH, L. A new memory monitoring scheme for memory-aware scheduling and partitioning. In Proceedings of the 8th International Symposium on High-Performance Computer Architecture (2002).
8. WOOD, T., SHENOY, P., VENKATARAMANI, A., AND YOUSIF,M. Black-box and gray-box strategies for virtual machine migration. In Proceedings of the 4th USENIX conference on Networked systems design and implementation (2007).
9. ZHURAVLEV, S., BLAGODUROV, S., AND FEDOROVA, A. Addressing shared resource contention in multicore processors via scheduling. In Proceedings of the 15th International Conference on Architectural support for programming languages and operating systems (2010).