Topology-aware Virtual Resource Management for Heterogeneous Multicore Systems

Jianmin Qian, Jian Li, Ruhui Ma
Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, China
Email: {hacker_qian, li-jian, ruhuima}@sjtu.edu.cn

***Abstract*—Virtualization technology consolidates multiple independent workloads on a single physical server with virtual resource management, which can result in a significant utilization improvement and energy saving. However, the management of virtual resource is becoming more and more challenging, due to the lack of accurate performance prediction model for the diverse applications' irregular resource access behaviors as well as the complicated Non-Uniform Memory Access (NUMA) server architecture. These challenges drastically affect the overall consolidation performance. This paper proposes *vTRMS*, a runtime Topology-aware virtual Resource Management Scheme for heterogeneous NUMA multicore systems. *vTRMS* can improve application performance based on the comprehensive online monitor of the application resource access behaviors as well as an accurate and platform-independent detected NUMA topology metric. Experiment results show that, compared with state-of-art approach, *vTRMS* can bring up an average throughput improvement of 28.3% and 36.2% on Intel and AMD NUMA machines respectively, when consolidating 32-VMs. At the same time, *vTRMS* only incurs a runtime overhead no more than 5%.**

I. Introduction

Server virtualization and workload consolidation have become the fundamental enabling technologies for large-scale cloud computing platforms, such as Amazon EC2 and Windows Azure. Pervasive user defined workloads are encapsulated in the Virtual Machines (VMs) to run on top of a physical server, and the automated virtual resources management policy is responsible to achieving the high resource utilization and low energy consumption. These workloads generate the massive and complex data interactions patterns, which incur the irregular memory footprint and high memory/network transmissions, such as online transaction processing (OLTP) [1] and in-memory computing [2]. These effects present a significant challenge for the resource management in virtualization systems to optimizing the system efficiency.

Meanwhile, Non-Uniform Memory Access (NUMA) machines comprise the majority of server architecture due to their high memory bandwidth and superior system scalability. A typical NUMA system consists of multiple nodes and each node is connected through a interconnect (e.g. Intel's QPI [3] or AMD's Hyper Transport [4]). In the NUMA systems, processor core gets much higher access to its local memory compared with the across-node remote access that incurs additional latency. Moreover, as the increasing number of nodes and cores are built into a NUMA system for higher computing capability, the underlying node-to-node interconnecting topology of the NUMA platform is becoming more heterogeneous, and the workload performance can be affected. For instance, prior study [5] has shown that the node-to-node interconnecting links of AMD 8-node Opteron NUMA architecture could be asymmetric, which fluctuates the bandwidth performance by more than 2x under the same distribution of thread and data across the nodes. Therefore, optimizing VM consolidation performance in a heterogeneous NUMA platform represents a key challenge in the cloud provider community.

Some previous research efforts [6], [7] have been devoted to improve the efficiency of workload consolidation with the NUMA-aware VM resource management policies. However, there are two major drawbacks for such approaches: First, the resource access behaviors of diverse application workload were modeled by an *offline profiling* approach, which cannot reflect the high dynamic application characteristics in runtime environment. Second, the underlying hardware topology is depicted by a symmetric distance metric for the node-to-node interconnect characteristics, but we argue the current NUMA platform topologies can be asymmetrically interconnected.

To this end, this paper presents ***vTRMS***, a Topology-aware virtual Resource Management Scheme for heterogeneous NUMA multicore systems. *vTRMS* not only monitors the virtual resource access behavior online but also adopts a novel accurate metric for predicting the underlying node-to-node interconnect performance. The dynamic runtime information and accurate metric are further employed to design a scheduling policy that improves the system efficiency by assigning the appropriate resources to a specific VM. Experimental results show that *vTRMS* has an average performance improvement of 28.3% on Intel platform and 36.2% on AMD platform respectively. Furthermore, *vTRMS* only increases runtime CPU overheads which no more than 5%. The major contributions of this paper can be summarized as follows:

- We design a platform-independent node-to-node latency detecting mechanism which can automatically and accurately measure the underlying interconnect latency.
- We propose *vTRMS*, a novel virtual resource management scheme to improve workload consolidation performance on NUMA multicore platform with an online virtual resources access behavior monitoring approach as well as the accurate node-to-node distance metric.
- We implement and evaluate our prototype on two different NUMA platforms with a set of cloud workloads, demonstrating significant performance improvements.

978-3-9819263-0-9/DATE18/©2018 EDAA

II. Related work

VM Resource Management: Many optimization approaches for VM resources management have been proposed to derive the efficiency benefits in server consolidation. Ye *et al.* proposed a profiling-based server consolidation framework which minimizes the number of Physical Machines (PMs) used in data centers [8]. Vasudevan *et al.* proposed a profile-based application assignment approach for more efficient data centers [9]. It builds realistic profiles from the raw data measured from data centers and then establishes a framework for application assignment. Yu *et al.* proposed a stochastic load balancing scheme which aims to provide probabilistic guarantee against the resource overloading with virtual machine migration [10]. These work focus on improving the data center resource utilization by designing a profiling-based framework to manage the VM clusters. Different from these approaches, we study the performance issues of the VM resource management policies for single PM to improve the consolidation efficiency.

NUMA-aware Optimization: Recent studies also highlighted various opportunities for enhancing workload performance in NUMA-based multicore systems. Tembey *et al.* focused on VM resource allocation on NUMA multicore systems to improve the efficiency and isolation of consolidated workloads [6]. Dashti *et al.* proposed *Carrefour* [11], an NUMA-aware thread and memory placement algorithm to avoid traffic hotspots and prevent congestion in memory controllers and on interconnect links. Diener *et al.* propose an automatic kernel-level memory affinity framework for NUMA systems [12], which manages both thread and data affinity according to the memory access behavior of parallel applications.

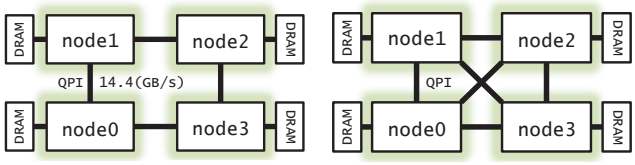
These researches focus on optimizing conventional data access overheads (e.g. remote memory access latency, memory controller congestion and interconnection congestion) on NUMA systems. However, they ignore to analyzing impacts of asymmetric underlying interconnects topology on applications performance. Prior work [5] has proved that the asymmetric NUMA node-to-node interconnects topology can vary the application performance by more than 2× in a bare metal environment. Accordingly, we also explore the impacts of the asymmetric NUMA topology on the VM resource management. Our *vTRMS* accounts for the asymmetric NUMA interconnect characteristic, and provides a more flexible resource management with the increasing number of consolidated VMs.

III. Background and Motivation

A. Heterogeneous Underlying NUMA Topology

Figure 1 describes two typical Intel NUMA multicore architectures. Each NUMA system consists of four nodes and each node connected via Intel's Quick Path Interconnect (QPI). The main difference between these two NUMA architectures is the number of per-node QPIs: each E5 processor has 2 way QPIs while E7 processor has 3 way QPIs. The transfer speed of these QPIs are both 14.4 GB/s and bidirectional, which make the interconnect topology symmetric.

Unlike the symmetric Intel interconnect links topology, the interconnect links of AMD NUMA architectures exhibit many



(a) Intel 4-node Xeon E5 (b) Intel 4-node Xeon E7

Fig. 1. Intel 4-node Ivy Bridge NUMA Architectures.

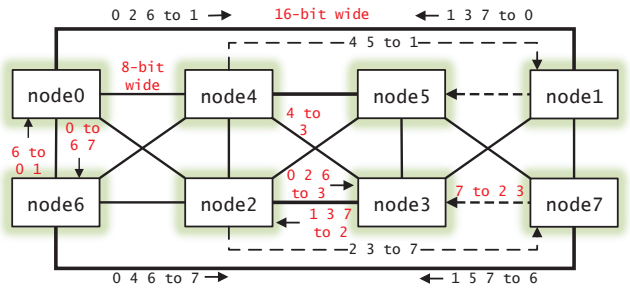


Fig. 2. AMD 8-node Opteron NUMA Machine.

disparities. Figure 2 shows the interconnect links topology of an AMD 8-node Opteron NUMA architecture. **(1)** Links between nodes have different bandwidths, as some are 16-bit wide (e.g. the link between node 0 and 7) and some are 8-bit wide (e.g. the link between node 0 and 4). **(2)** Links have different transfer directions, while some are bidirectional, others are unidirectional. For instance, node 7 sends requests directly to node 3, but node 3 routes its answers via node 1. **(3)** Links are shared different; some are shared by two nodes, but others are shared by multiple nodes. For instance, the link between node 4 and 3 is only used by these two nodes, while the link between 2 and 3 is shared by nodes 0, 1, 2, 3, 6 and 7. We explore the impacts of these asymmetric topology features on VM resource access performance in the following section.

B. Motivations

*1) **Offline Profiling to Characterize Behavior**:* Previous works [8][9][10] characterize the virtual resource access behaviors by using an *offline profiling* approach. For instance, prior work [13] attempt to capture the workload memory access behavior in NUMA platform by profiling the host system file *numa_maps*. While such offline profiling approach might be useful in some scenarios as well as not involving too much runtime overheads, there are two major drawbacks to such an approach. First, it might not always be practicable to profile the host system. For instance, in a cloud service provider such as Amazon EC2 where the host system cannot expose to any user for the sake of system security, it is impossible to profile applications offline. Second, even when the workload can be profiled offline, due to changing inputs at the workload execution phase, the memory access behavior might be different compared to when the offline profiling was performed. Hence, such an offline profiling approach shows poor applicability.

*2) **Inaccurate Distance as Model Metric**:* Most NUMA-aware resources management policies, including the commercial products [13], use the node-to-node distance metric, such as the System Locality Information Table (SLIT) [14] to model the underlying hardware topology. As shown in the Table I, the SLIT distance is a symmetric matrix by assuming that the forward and reverse distance between two nodes are same. Therefore, it is unsuitable to describe the asymmetric physical interconnect links, as we described in the section III-A. We would like emphasize that the metric values are very inaccurate to depict the physical interconnect performance.

To this end, we run the `STREAM` benchmark inside a VM with different virtual CPU (vCPU)/memory placements on the asymmetric AMD NUMA platform and Table II shows the results. The first row indicates the node index where the VM vCPU is mapped and the first column indicates the node index where that VM memory was allocated. See in Table II that: **(1)** vCPU on node 0 can access the memory on node 7 with 3.7 GB/s bandwidth, while vCPU on node 7 accesses the memory on node 0 with 2.0 GB/s bandwidth; **(2)** vCPU on node 7 can access its memory on node 3 at 3.1 GB/s bandwidth while vCPU on node 3 accesses the memory on node 7 with 4.3 GB/s bandwidth; **(3)** vCPU on node 4 accesses the memory on node 3 with 3.9 GB/s bandwidth, while vCPU on node 2 accesses the memory on node 3 with 5.2 GB/s bandwidth. Observe that the VM bandwidth performance also can be affected by the asymmetric matters. Consequently, *it is inappropriate to use the SLIT distances as bandwidth performance model metric*.

TABLE I. SLIT DISTANCE OF AMD MACHINE.

Node	0	1	2	3	4	5	6	7
0	**10**	16	16	**22**	16	**22**	16	**22**
1	16	**10**	**22**	16	16	**22**	**22**	16
2	16	**22**	**10**	16	16	16	16	16
3	**22**	16	16	**10**	16	16	**22**	**22**
4	16	16	16	16	**10**	16	16	**22**
5	**22**	**22**	16	16	16	**10**	**22**	16
6	16	**22**	16	**22**	16	**22**	**10**	16
7	**22**	16	16	**22**	**22**	16	16	**10**

TABLE II. BANDWIDTH (GB/s) OF AMD MACHINE.

Node	0	1	2	3	4	5	6	7
0	5.6	5.2	4.4	3.9	3.0	2.1	2.9	**2.0**
1	4.6	5.7	3.9	4.3	2.1	3.0	2.0	2.9
2	3.0	3.8	5.7	5.1	4.3	2.0	3.0	2.1
3	2.1	4.4	**5.2**	5.6	**3.9**	3.1	2.1	**3.1**
4	4.3	2.0	3.1	2.1	5.5	5.2	3.1	3.8
5	3.8	3.0	2.1	3.1	5.1	5.6	2.1	4.2
6	4.1	1.9	3.0	3.8	3.1	2.0	5.6	4.6
7	**3.7**	2.9	2.0	**4.3**	2.1	3.0	5.2	5.5

Our goal, in this work, is to design an NUMA-aware VM resources management scheme that improves workloads consolidation performance by adopting an online profiling approach as well as an accurate interconnect distance metric.

IV. DESIGN

A. Overview

Figure 3 gives an overview of *vTRMS*. *vTRMS* consists of three components: an ***Online Behavior Monitor*** that monitors each VM's runtime resource access behavior periodically; an ***Underlying Topology Detector*** that detects the NUMA host interconnect topology to provide an accurate distance metrics for the scheduler; and a ***Runtime vCPU Scheduler*** migrates VM's vCPU according to the monitoring and detecting data.

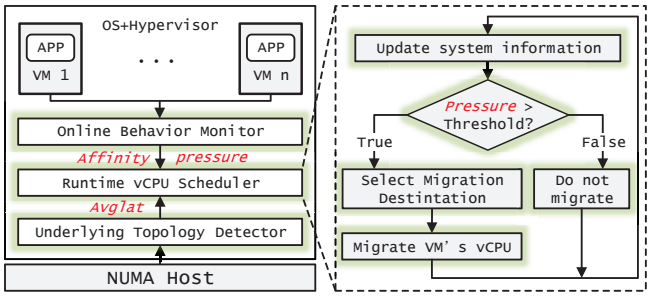


Fig. 3. Workflow of *vTRMS*.

B. Online Behavior Monitor

Online behavior Monitor module is responsible to dynamically monitoring the VM resources (vCPU and memory) behaviors to predict the workload performance. In order to make accurate scheduling decisions, the scheduler needs to know the runtime characteristics of workloads. To this end, *Online Behavior Monitor* uses the Linux performance monitoring tool *perf* to collect and analyze the VM's vCPU and memory access characteristics. The collected hardware performance events are listed in the Table III and monitoring behaviors include two aspects: (1) The VM vCPU's ***Last Level Cache (LLC) access pressure***, which represents the current shared resource access overhead of VM's vCPU; (2) The ***vCPU-to-node access affinity***, which represents the affinity between VM's vCPU and its memory on each NUMA node. These two behavior features are explained in detail in what follows:

TABLE III. HARDWARE PERFORMANCE EVENTS.

Hardware Events	Description
LLC_MISSES	Last level cache misses
INSTRUCTION_RETIRED	Retired instructions
UNC_QMC_NORMAL_READS	Memory reads
UNC_QMC_WRITES	Memory writes

vCPU's LLC access pressure: we use the number of last level cache (LLC) misses as the effective metric to quantify the shared resource access overhead. Concretely, it exhibits whether the VM suffers from the remote resource access overhead. Different with the previous work, we consider the VM's vCPU access behavior by calculating LLC misses per instruction (MPI) as the pressure. MPI is more suitable to characterize the access overhead on the specific hardware machine.

As shown in the Equation (1), Where the LLC_Misses is the number of LLC_MISSES and the $Instruction\_Retired$ is the number of INSTRUCTION_RETIRED of the VM vCPU.

$$Pressure = \frac{LLC\_Misses}{Instruction\_Retired} \quad (1)$$

vCPU-to-node access affinity: Since the physical latency between each node is constant, the access penalty is determined by the frequency of VM's vCPU access its memory on the node. Thus we introduce the vCPU-to-node access affinity, which denotes access behavior between VM's vCPU and memory on each node. For a given vCPU, its affinity to node i, $Affinity_i$, is defined as the proportion of its memory access on node i ($NodeAcc_i$) over the total number of memory access $\sum_{i=1}^{N} NodeAcc_i$, where N is the number of nodes in NUMA system. The number of VM memory access to the node i in the monitoring period is calculated by adding the values of two related hardware performance events UNC_QMC_NORMAL_READS and UNC_QMC_WRITES, which reflect the number of reads and writes to the node i attached memory bank, respectively.

$$Affinity_i = \frac{NodeAcc_i}{\sum_{i=1}^{N} NodeAcc_i} \quad (2)$$

C. Underlying Topology Detector

Underlying Topology Detector is a platform-independent node-to-node latency detecting scheme that measures the data access latency between each NUMA node automatically. This node-to-node interconnect latency is employed as the distance metric between nodes, since it is correlated with the access latency between each node [5]. Notice that the underlying distance topology is static that only need to detect once at the system initialization phase. The detected value will be saved into a matrix (***Avglat***) which can replace the traditional SLIT distance table to reflect the accurate interconnect topology for performance influence.

Given a server with the number of NUMA nodes n, the latency between node i and j is obtain as follows: we first allocate a buffer in node j via the `numa_alloc_onnode()` function provide by *libnuma* library; Second, we create a read thread and bind it to a processor on node i. Third, we record the CPU cycles at the beginning of buffer reading (with the `RDTSC` instruction that returns the number of CPU cycles that occurred since system reset). Forth, when the buffer reading ends, we get the number of ended CPU cycles and calculate the total cycles during the buffer reading process as equation (3). Finally, the average access latency between each NUMA node is calculated as Equation (4), where $time\_per\_cycle = \frac{1}{cpu\_freq}$ denotes an actual CPU cycle time and $total\_read\_num$ denotes the read number during buffer reading. After calculating the latency between each node, a $n \times n$ latency matrix is obtained to be ***Avglat***.

$$total\_cycles = end\_cycles - start\_cycles \quad (3)$$

$$latency = \frac{total\_cycles * time\_per\_cycle}{total\_read\_num} \quad (4)$$

D. Runtime vCPU Scheduler

With the knowledge of the runtime memory access behavior and the node-to-node interconnect topology, the vCPU scheduling algorithm is responsible to carry out an efficient resource management that consists of the following steps:

Step 1: Whenever the scheduler periodically selects the VM's vCPU for migration, it first updates the system-wide information with new monitoring data. However, determining the update period is very important. Too frequent updates will increase system overhead and low frequent updates cannot make right migration decision. We empirically set the update period to 1 second for good balance of overhead and efficiency.

Step 2: After updating per VM runtime information, the scheduler identifies which VMs should be migrated by analyzing each VM's vCPU LLC access pressure ($pressure$). To decide whether a VM suffers from the LLC access overhead or not, a threshold of the $pressure$ should be determined experimentally. We observed that the threshold of 10000 worked well on the system we evaluated, and it is not very sensitive to different workloads. Once the $pressure$ value of the VM exceed the threshold, the VM will be added to candidate queue for scheduling. We also set a timestamp flag (`T_flag`) for each VM to avoid unfair scheduling. If the VM waiting in the candidate queue for a long time ($\geq$ 180s) to finding a migration destination, we increase the scheduling priority of the VMs according to the `T_flag`.

Step 3: Once deciding to migrate the VM's vCPU, the migration destination will be a core on a target NUMA node. Thus the scheduler first identifies which node the VM's vCPU should migrate to. To this end, the scheduler calculates the Average Access Latency (AAL) by multiplying the values in the vCPU-to-node access affinity vector $Affinity$ and node-to-node latency matrix $Avglat$ together. For instance, the current VM's vCPU is mapped on node 0 and $Affinity$ saves the VM vCPU-to-node access affinity between node 0 and each node, $Avglat$ provides the physical access latency from node 0 to each node. As shown in the Equation (5), since the higher $Affinity$ value is better and the lower $Avglat$ is better, we get the reciprocal of $Avglat$ and return the node with maximal AAL value.

$$AAL = Affinity * \frac{1}{Avglat} \quad (5)$$

After identified the target node, the scheduler selects a core on this node to migrate the VM's vCPU. For the purpose of balancing system load, a less-busy core (lowest per-core CPU utilization) is selected as the final destination.

V. Evaluation

This section presents the evaluation results and analyzes of our prototype. We first show the $Avglat$ latency distance is a reliable architectural metric. We then show the effectiveness of our *vTRMS* with a set of cloud workloads. We compare the performance of our *vTRMS* with current Linux resource management tools, including ***numad*** and ***AutoNUMA***. ***numad*** is an automatic NUMA affinity management daemon, which

dynamically improves workloads bandwidth performance with SLIT distance [13]. ***AutoNUMA*** is a NUMA-aware approach that keeps processes and their memory together on the same node [15]. We finally characterize *vTRMS* runtime overheads.

A. System Configuration

All experiments were conducted on two NUMA platforms: **Platform A: Intel with 4 NUMA nodes** is a server equipped with Intel Xeon E5-4610 processors. The 4 nodes are interconnected with 2-ways QPI that have a communication speed of 7.2 GT/s and 32GB DDR3 memory attached to each node. Each processor with 8 cores runs at 2.0GHz.
Platform B: AMD with 8 NUMA nodes is a server equipped with AMD Opteron 6376 processors. It features 8 nodes interconnected with 6.4 GT/s Hyper Transport 3.0 links. Each processor is with 16 cores runs at 2.3GHz and with a 16GB DDR3 memory attached to each node.

Cloud Workloads: The ***httperf*** with *Apache* that measures the number of http requests per second; ***Memcached*** with *Memslap* is a distributed in-memory key-value storage to measure the online requests; The ***YCSB*** with *MYSQL* database that evaluates data reading, writing and updating performance. The performance comparison of each workload is under four scheduling choices: (*numad*, *Autonuma*, *vTRMS with SLIT* and *vTRMS with Avglat*). Evaluation results are average values of ten times execution, and all values are normalized to the performance of *numad*.

B. The Accuracy of the Avglat

The first experiment evaluates the accuracy of our *Avglat* values. We compare the detected values with the actual values obtained from the standard latency benchmark *LMbench*, and report the **Mean Absolute Percentage Error (MAPE)** which is defined as Equation (6). Where the Lat_{actual} is the value measured by the *LMbench* and the Lat_{detect} is the detected *Avglat* latency values.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{Lat^{i}_{actual} - Lat^{i}_{detect}}{Lat^{i}_{actual}}\right| \quad (6)$$

Table IV shows the MAPE values on the AMD platform. The MAPE is in range of 0.1%-6.5% and the average MAPE is 3.3%. In some case (such as node 0 to node 3, node 2 to node 1), the MAPE value is close to 0, which means the $Avglat$ values are very closed to the benchmark values. We also evaluated the MAPE values on our Intel platform, and the average MAPE is 2.9%. Based on the above experiment data, we conclude that our $Avglat$ latency matrix can accurately measures the node-to-node interconnect latency.

C. Cloud Workloads Results

Figure 4 shows the performance comparisons among different policies on the Intel **platform A**. For all the workloads, *vTRMS with Avglat* always achieves the highest performance among the four mechanisms. For example, when consolidated 32 VMs with *YCSB*, *vTRMS with Avglat* outperforms the baseline *numad* by 32.4% and *Autonuma* by 16.1%. This is because *vTRMS with Avglat* also provides flexible awareness both on workloads runtime access behaviors and the underlying hardware characteristics. For *vTRMS with SLIT*, it performs close to *vTRMS with Avglat*. As we described in the section III-A, the underlying interconnect links of our Intel platform is symmetric, thus *vTRMS with SLIT* and our *vTRMS with Avglat* performed consistently. For the workloads with high memory bandwidth consumption, our *vTRMS with Avglat* improves the overall performance more obviously. For example, when consolidated 32 VMs with *Memcached*, *vTRMS with Avglat* outperforms 40.7% than the default *numad*. Since *vTRMS with Avglat* consider the dynamic bandwidth demand of workload, when consolidating VMs with the such workloads with high memory bandwidth consumption, the performance improvement of *vTRMS with Avglat* is more significant.

TABLE IV. MAPE (%) OF THE AMD PLATFORM.

Node	0	1	2	3	4	5	6	7
0	3.2	5.1	3.4	**0.1**	3.8	1.1	1.9	2.8
1	4.8	4.1	3.9	4.3	2.5	4.7	1.1	2.4
2	3.0	**0.3**	2.7	6.1	5.3	3.0	4.7	**0.5**
3	2.1	**0.8**	4.4	2.1	3.7	6.5	5.1	4.9
4	4.3	2.0	3.1	**0.8**	2.5	5.2	3.1	3.8
5	5.8	4.5	1.1	1.7	3.2	4.6	**0.4**	3.1
6	4.1	6.2	**0.9**	1.4	**0.6**	3.9	5.9	3.7
7	1.1	2.9	5.1	3.2	4.3	3.1	6.3	3.2

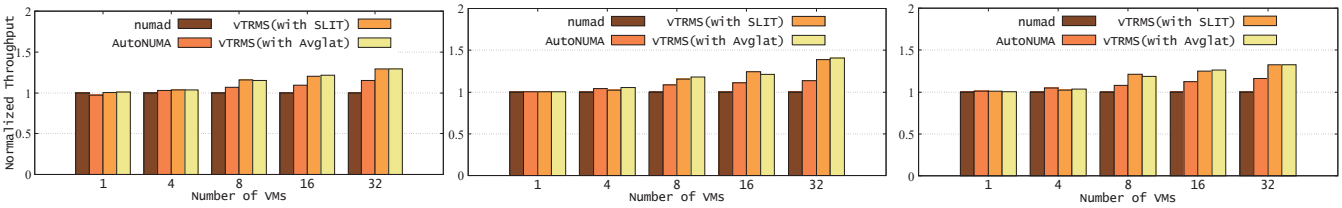
Figure 5 shows the experiment results on the AMD **platform B**. Comparing with the Intel platform, we observe that *vTRMS with Avglat* performs much higher than *vTRMS with SLIT*. For example, when consolidating 32 VMs with the YCSB, *vTRMS with Avglat* achieves 38.4% higher performance than the default *numad*, while the *vTRMS with SLIT* achieves 20.9% higher. Note that the AMD platform features 8 nodes and its underlying interconnect links are asymmetric, whereas the SLIT provides a symmetric distance matrix and cannot reflect this asymmetry topology. This proves that, with our *Avglat* latency, *vTRMS* can accurately model the system overall performance for the asymmetric AMD platform.

D. Overheads

Since *vTRMS* runs as a daemon process in the user level, it inevitably brings overheads. The main overhead is attributed to the following aspects: collecting the VM runtime information, detecting the underlying topology, as well as migrating vCPUs among nodes. Figure 6 shows the total increasing CPU usage of *vTRMS* with different number of consolidated VMs. First, when consolidating with small number of VMs (1 or 4), the increased CPU usage is no more than 2%. Since fewer VMs will result in lower shared resource accesses overhead. Second, Memcached always incurs a little higher CPU usage increase due to its higher bandwidth consumption, but the difference between each workload is unobvious (within 1%).

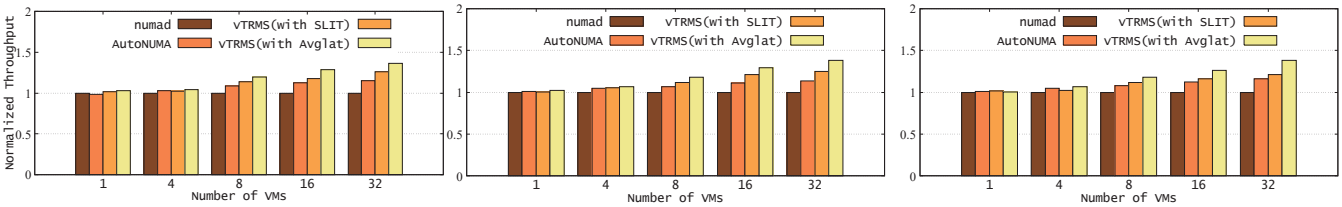
VI. Conclusion

This work explores the limitations of current virtual resource management policies in heterogeneous NUMA multicore sys-



(a) httperf (b) Memcached (c) YCSB

Fig. 4. Workload performance improvement on Intel platform.



(a) httperf (b) Memcached (c) YCSB

Fig. 5. Workload performance improvement on AMD platform.

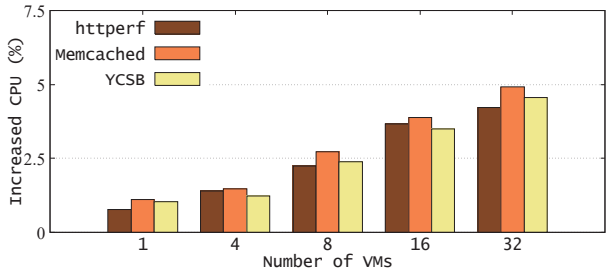


Fig. 6. Increased CPU usage.

tems. To address these issues, we presented and implemented *vTRMS*, a topology-aware VM resource management approach to optimize cloud workloads consolidation performance in NUMA multicore system. Experiments with different workloads form both Intel and AMD platforms demonstrate an average improvement of 28.3% and 36.2% respectively on performance in comparison with the Linux default scheduler.

ACKNOWLEDGMENT

This work was supported in part by the National Key Research Development Program of China (2016YFB1000502), National Natural Science Funds for Distinguished Young Scholar No.61525204. Jian Li is the corresponding author.

REFERENCES

[1] H. Plattner, "The impact of columnar in-memory databases on enterprise systems: Implications of eliminating transaction-maintained aggregates," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1722–1729, Aug. 2014.

[2] T. Jiang, Q. Zhang, R. Hou, L. Chai, S. A. Mckee, Z. Jia, and N. Sun, "Understanding the behavior of in-memory computing workloads," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2014, pp. 22–30.

[3] B. Mutnury, F. Paglia, J. Mobley, G. K. Singh, and R. Bellomio, "Quickpath interconnect (qpi) design and analysis in high speed servers," in *19th Topical Meeting on Electrical Performance of Electronic Packaging and Systems*, Oct 2010, pp. 265–268.

[4] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway, "The amd opteron processor for multiprocessor servers," *IEEE Micro*, vol. 23, no. 2, pp. 66–76, March 2003.

[5] B. Lepers, V. Quema, and A. Fedorova, "Thread and memory placement on NUMA systems: Asymmetry matters," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, 2015, pp. 277–289.

[6] M. Liu and T. Li, "Optimizing virtual machine consolidation performance on numa server architecture for cloud workloads," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, June 2014, pp. 325–336.

[7] P. Tembey, A. Gavrilovska, and K. Schwan, "Merlin: Application- and platform-aware resource allocation in consolidated server systems," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SOCC '14. New York, NY, USA: ACM, 2014, pp. 14:1–14:14.

[8] K. Ye, Z. Wu, C. Wang, B. B. Zhou, W. Si, X. Jiang, and A. Y. Zomaya, "Profiling-based workload consolidation and migration in virtualized data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 878–890, March 2015.

[9] M. Vasudevan, Y.-C. Tian, M. Tang, and E. Kozan, "Profile-based application assignment for greener and more energy-efficient data centers," *Future Generation Computer Systems*, vol. 67, no. Supplement C, pp. 94 – 108, 2017.

[10] L. Yu, L. Chen, Z. Cai, H. Shen, Y. Liang, and Y. Pan, "Stochastic load balancing for virtual resource management in datacenters," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[11] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth, "Traffic management: A holistic approach to memory placement on numa systems," *SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 381–394, Mar. 2013.

[12] M. Diener, E. H. M. Cruz, P. O. A. Navaux, A. Busse, and H. U. Hei, "kmaf: Automatic kernel-level management of thread and data affinity," in *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, Aug 2014, pp. 277–288.

[13] https://linux.die.net/man/8/numad.

[14] D. Schmidl, C. Terboven, and D. an Mey, "Towards numa support with distance information," in *Proceedings of the 7th International Conference on OpenMP in the Petascale Era*, ser. IWOMP'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 69–79.

[15] J. Corbet, "Autonuma: the other approach to numa scheduling," *LWN.net*, 2012.