vNUIOA: Optimizing Non-Uniform Access in Virtualized Environment for Multicore Systems

ABSTRACT

Current data center servers are equipped with multicore processors and high speed network adapter simultaneously to improve the cloud application performance. With the trends of increasing CPU cores in a single physical machine, the access uniformity is also losing. Previous research mainly focused on optimizing remote memory access in multicore systems. However, in this paper, we show that, with the speed of I/O devices getting faster and faster, non-uniform I/O access also undermines multicore systems performance. Consequently, the non-uniform memory and I/O access aggravates overall system degradation. Virtualization also brings an additional abstraction of underlying topology and makes the optimization more challengeable.

In this paper, we first analyze the non-uniform memory and I/O access comprehensively in the virtualized environment. Then we present a runtime affinity driven virtual machine management framework, *vNUIOA*, which dynamically optimizes both non-uniform memory and I/O access based on the hardware topology. We have implemented *vNUIOA* in the KVM virtual machine monitor. Experiment results from real multicore machines show that *vNUIOA* has an average performance improvement of 32.9% (max up to 39.4%) on single type workloads and 28.2% (max up to 45.4%) on mixed type workloads comparing with KVM default NUMA aware scheduler. *vNUIOA* also improves performance with no more than 5% variation.

1. INTRODUCTION

Modern computer systems employ multicore processors, which could significantly improve the performance of cloud computing infrastructures. Driven by the rapid growth of big data applications, current cloud computing servers are also equipped with high speed network adapters to efficiently transfer data. However, the non-uniform access feature in the multicore systems, such as Non-Uniform Memory Access (NUMA)[], severely limits the performance of these powerful platforms. Several performance degradation was observed due to poor management in the NUMA architecture[]. Existing work focuses on optimizing of data locality to avoid massive remote access [1, 2, 3], and NUMA aware virtual machine scheduling to improve performance for cloud workloads[].

In the NUMA architecture, each CPU socket accesses directly attached memory banks faster than the remote ones []. However, directly attached devices are not limited to memory, I/O devices such as high speed Ethernet network adapter [] also directly connect to the CPU socket through the interconnect bus. When CPU cores access I/O devices remotely, latency will increase and inter-node bandwidth is consumed quickly. Since the speed of network interface card (NIC) get faster [], this Non-Uniform I/O Access (NUIOA)[] also becomes a bottleneck of today's cloud based high performance computing.

A large body of recent work has realized the impact of non-uniform I/O access in cloud environment. Xen supports IONUMA [] by running VMs on those nodes which I/O devices are directly attached to. OpenStack plans to do I/O based NUMA scheduling [] to achieve a high performance, low latency system for Network Function Virtualization (NFV)[] workloads. VMware [] also implements NUMA aware I/O scheduler for virtualized systems. All these work expects to improve network performance by optimizing the affinity between virtual machines and I/O devices.

Although previous studies[] presented several affinity optimization models. Unfortunately, these models are not suitable for current cloud environment with high performance networking for following reasons:

1) Existing affinity models mainly focus on optimizing the remote memory access or remote I/O access. The analysis of relationship between non-uniform memory and I/O access is not enough comprehensive, and result in inefficiency of previous model. Since almost the cloud space applications with huge amount of data computing and transferring, current high performance computing systems must optimize these two non-uniform access simultaneously.

2) Many traditional NUMA performance models use the hop distance to introduce the affinity between two devices. However, hop distance is not accurate for affinity modeling, especially for the I/O performance due to: Firstly, the hop distance contains less details about the underlying topology which can not be used in the accurate performance measurement. Secondly, hop distance is recorded as a constant value in the system. When the performance of the system changes in real time, the hop distance can not reflect this change and cause the

performance prediction inaccurate.

3) Furthermore, previous work only optimize the non-uniform I/O access by running virtual machines on those nodes which I/O devices directly attached to. This optimization ignore the VMM level nontransparent management and result in optimizing inefficient. Because this static method can only ensure I/O access locally in the initialization. When running large amount of VMs in these nodes will incur high CPU loads, and the system load balancing mechanism migrates VMs to other nodes.

In this paper, we present a runtime affinity driven virtual machine management framework, *vNUIOA*, which takes both non-uniform I/O and memory access into consideration in the virtualized systems. *vNUIOA* first detects the non-uniform I/O and memory access behavior since the booting of each virtual machine, *vNUIOA* analyses hardware topology based on the host machine at the same time. After that, *vNUIOA* use these integrated information in the calculation of optimal affinity. Finally, with the optimal affinity combination , *vNUIOA* periodically makes the decision of resources management according to the runtime system load.

We evaluated our *vNUIOA* system on a real Intel machine with different type workloads, *vNUIOA* shows high efficiency. For single type workload VM, *vNUIOA* has an average performance improvement of 32.9% (max up to 39.4) compared with KVM default scheduler. For mixed type workload VM, *vNUIOA* improves performance by average to 28.2% (max up to 45.4%). With the number of VM increases, *vNUIOA* continuously improves performance with no more than 5% variation. The contributions of this paper include:

- Comprehensive evaluations on performance degradation due to non-uniform memory and I/O access in multicore systems, address the main factors that impact network performance in cloud environment.
- A detailed analysis of affinity classes and use more accurate metrics for the optimal affinity modeling with taking these factors into consideration at the same time.
- A design of runtime affinity driven resources management framework with the consideration of online system load.

This paper is organized as follows: Section 2 provides an overview of non-uniform access architecture and motivation for this research. Section 3 analyzes affinity in current systems and provides an optimal affinity model. Sections 4 and 5 present our design and implementation. Section 6 evaluates our prototype and compares with existing methods. Section 7 discusses related work and Section 8 concludes this paper.

2. BACKGROUND AND MOTIVATION

In this section, we begin with a brief introduction to non-uniform access architecture in multicore systems. Then we show performance degradation due to the non-uniform access in high performance network environment. Lastly, we point out that virtualization also aggravates the performance degradation.

2.1 Non-Uniform Access in Multicore Systems

Non-Uniform Memory Access: Traditional multicore processor shares one memory controller. With the number of cores per socket increasing, one memory controller results in high memory controller contention and bandwidth competition. In order to improve the utilization of multicore system, each CPU socket has been designed with its own integrated memory controller (IMC) which have faster access to local memory bank than the remote ones. This non-uniform memory access characteristic fully takes advantage of multicore system and high speed memory bandwidth.

Non-Uniform I/O Access: Besides the non-uniform memory access, NUMA architecture also results in non-uniform access to other devices. One of the most important is the I/O devices, because I/O devices are also directly connected to one or more nodes in the multicore system. Local cores and devices have privilege access to I/O devices but remote cores must use interconnection to transfer data to the I/O devices, resulting in extra inter-node bandwidth occupation and access latency. This non-uniform I/O access is called Non-Uniform I/O Access (NUIOA)[]. AMD processors have exhibited NUIOA effects when AMD introduced the OPTERON processor in 2003 [], Intel processors have begun to appear NUIOA characteristic when the advent of *sandy-bridge* architecture in 2011 []. Figure 1 shows an ubiquitous four-sockets NUIOA architecture based on the Intel processors. Each socket consists of eight cores that share the last level cache (L3 cache), memory controller and a physical memory bank. sockets link with each other via a high-speed, point-to-point interconnection such as Intel Quick Path Interconnect (QPI)[]. Network adapter is directly attached to the socket 0 with the Intel I/O controller [], consequently, data from NIC transfers remotely to other sockets and increases access latency and bandwidth consumption.

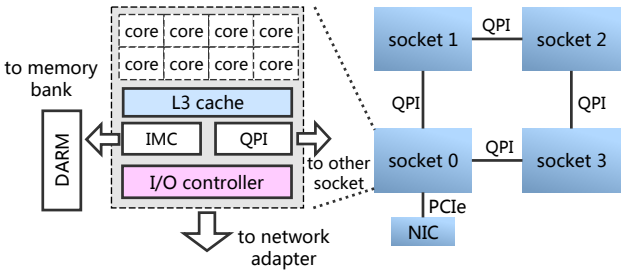


Figure 1: Asymmetric I/O access architecture with four 8-cores Intel processors.

2.2 The Impact of Non-Uniform Access on Performance

To measure the performance degradation due to non-uniform access in multicore systems, we conduct a set of experiments. Since the non-uniform memory access is a widely known problem, we first evaluate the network performance degradation of non-uniform I/O access independently. Then we combine non-uniform memory and I/O access and study the factors of performance degradation.

2.2.1 *Moving from 10GbE to 40GbE*

Previously, since the speed of 10 Gigabit Ethernet adapter is much slower than the bandwidth of memory and interconnection, the 10 GbE adapter can be taken fully advantage of and the main system bottleneck is non-uniform memory access. With the high performance NIC such as Mellanox 40 and 56 Gigabit Ethernet adapter being broadly adopted in the data center, the data transfer speed of current network becomes faster and faster and results in shifting of system bottleneck from non-uniform memory access to non-uniform I/O access in the multicore systems. Finally, without consideration of the non-uniform I/O access in high speed networking environment, the performance of data center servers degrades significantly.

2.2.2 *Evaluation of Non-Uniform I/O Access*

To study the impact on performance of asymmetric I/O access independently, the vCPU and memory should be mapped to the same node to ensure local memory access. We select Netperf benchmark [] to provide network bandwidth testing between two Intel host on a network. The configuration details of our test hardware are shown in the section 6. we simply run a number of VMs based on the asymmetric I/O access architecture by varying vCPU and memory mappings together among the multiple sockets. Figure 2 shows a four-sockets Intel Ivy Bridge NUMA architecture with high speed network interface card. Firstly, we bind the vCPU of the VM to a specific core on the socket 0 and map all the memory of the VM to the memory bank attached to socket 0, then we change the VM vCPU binding to the core on the socket 2 and map memory of VM to the socket 2. Traditionally, these two kinds of mapping strategy ensure the VM vCPU thread is locally access to the VM memory, so the affinity between VM vCPU and memory can achieve best. However, we observe a huge I/O performance difference between this two kinds of VM vCPU and memory mapping strategy.

Figure 3 gives the performance difference of these two kinds mapping strategies with Intel 10 Gigabit Ethernet adapter and Mellanox 40 Gigabit Ethernet adapter. As for the Intel 10 Gigabit Ethernet adapter, we observe a little throughput improvement (average 3%) since the bandwidth of this network adapter can be fully occupied. But for 40 Gigabit network adapter, the speed of network adapter is not bottleneck. I/O throughput of first kind of mapping (VM mapping 1)is average 52% (max up to 101.6%)higher than the I/O throughput of second kind of mapping (VM mapping 2) with different message sizes. The main reason is that when transfer data with 40 Gigabit network adapter, the second kind of mapping need to use two hops interconnection between sockets, this increases the QPI bandwidth congestion and memory access latency. Finally, network I/O throughput will be declined.

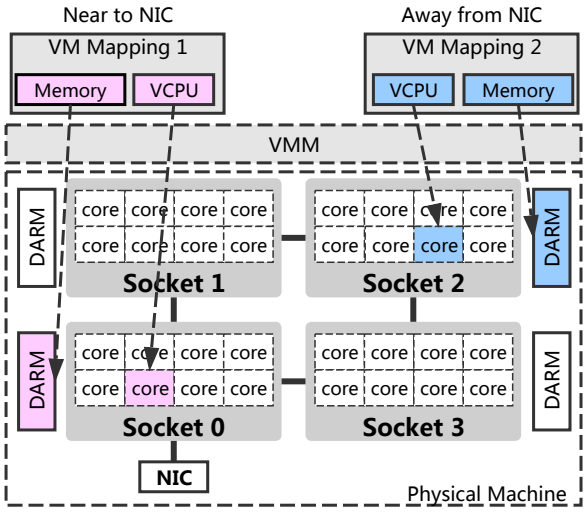


Figure 2: Different VM mapping strategies based on the asymmetric I/O Access architecture.

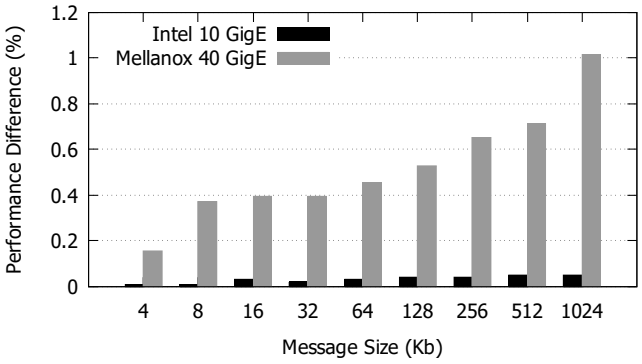
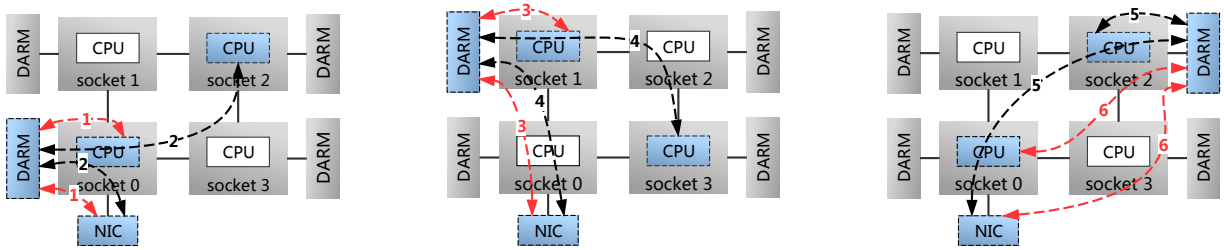


Figure 3: Performance degradation of different VM mapping strategies (VM mapping 2 relative to VM mapping 1) by using Intel 10 Gigabit Ethernet adapter and Mellanox 40 Gigabit Ethernet adapter.

2.2.3 *Evaluation of Non-Uniform I/O Access and Memory Access*

To study the factors of performance degradation in more complex scenarios, we consider the non-uniform I/O access and remote memory access simultaneously. We use a both data and computing intensive benchmark YCSB [] to evaluate the performance. Figure 4 shows six testing scenarios to characterize these two factors. In the Figure 4(a), we first map all the VM memory to the memory bank attached to the socket 0 and bind VM vCPU to the cores on socket 0 (mapping 1), then



(a) Mapping strategies 1 and 2 (b) Mapping strategies 3 and 4 (c) Mapping strategies 5 and 6

Figure 4: Six scenarios for evaluating asymmetric I/O access and memory access

we change the vCPU banding to socket 2 (mapping 2). This scenario ensures all the VM memory in the I/O device directly attached socket and vCPU access to memory locally or remotely. Similar in the Figure 4(b), we consolidate all the VM memory on the socket 1 and bind the vCPU respectively to socket 1 (mapping 3) and 3 (mapping 4). This scenario tests the performance of NIC to memory 1-hop distance and vCPU access memory locally or remotely. Figure 4(c) shows the NIC to VM memory 2-hops distance and memory access locally (mapping 5) or remotely (mapping 6). Note that we do not consider the VM vCPU and memory mapping to socket 3 because of that in our test bed socket 1 and socket 3 are symmetrical. Table 1 gives the details of six mapping strategies.

Mapping strategies	NIC	memory	vCPU
1	node 0	node 0 (local)	node 0 (local)
2		node 0 (local)	node 2 (2-hop)
3		node 1 (1-hop)	node 1 (local)
4		node 1 (1-hop)	node 3 (2-hop)
5		node 2 (2-hop)	node 2 (local)
6		node 2 (2-hop)	node 0 (2-hop)

Table 1: Six evaluation strategies with different vCPU and memory mapping.

Figure 5 gives the evaluation results of these six scenarios. We use the performance of mapping strategy 1 as the normalized performance. comparing the performance of mapping strategies 1 and 2, we can see that remote memory access degrade 18% performance against the local access. When vCPU accesses memory locally and NIC accesses memory remotely, we observe a 34% performance decrease from the comparing of mapping strategy 1 and 5. Lastly, when vCPU accesses memory and NIC accesses memory both remotely, we can observe a 59% performance degradation on throughput by comparing the mapping strategy 1 and 6. From this evaluation, we find that the total degradation(52%) caused by only memory remote access (18%) and only I/O remote access (34%) is less than the degradation of both memory and I/O remote access (59%). We can conclude that with non-uniform access architecture, the

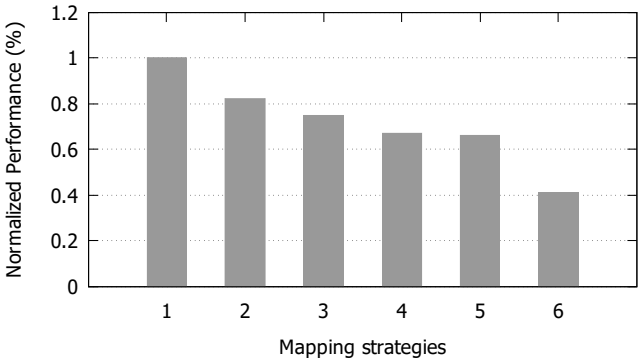


Figure 5: Evaluation results of six mapping strategies based on the asymmetric I/O and memory access architecture.

impacts on performance is more complicated.

Summary: Due to the impacts of non-uniform I/O and memory access, the performance decreases significantly in multicore systems. We should both optimize these non-uniform access rather then optimize single factor. However, in the virtualized environment, the nontransparent resources management may causes both memory and I/O remote access and degrades performance. We will discuss this in the next section.

2.3 Complications in Virtualized Systems

Virtualization poses additional abstraction of the underlying topology and the mapping of VM vCPU and memory in the virtualized environment is more complex. This complexity speeds up the performance degradation seriously. Current VMMs try hard to keep VM memory and vCPU as local as possible (like Xen and VMware ESXi), but mapping between virtual and machine resources impact by other system mechanisms. These mechanisms may disturb the distribution of VM vCPUs and memory and cause the remote access. According to the white paper of VMware [], the main reason of chaotic mapping in the virtualized system can be listed as follows:

1. The heterogeneity of the virtual machine will impact the resources mapping. Different kinds of applications running on the VM change the resources

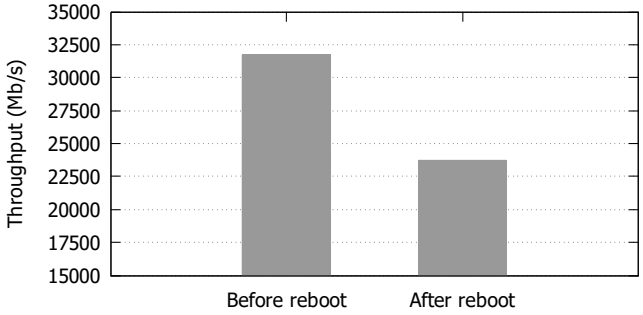


Figure 6: Performance degradation due to virtualization.

requirement. For example, memory intensive applications running in the virtual machine will result in the VM need lager size of memory, and these large amount of VM memory may distribute across multiple nodes and incur remote access .

2. The load balancing mechanism causes the VM memory and vCPU threads migration. Guest OS shutdown and reboot both trigger load balancing in the host OS, the prior mapping relationship will be changed. If VM memory and vCPU have been migrated to the node and cause remote access, the performance can be affected.

In order to verify these factors degrading network performance. We used different kinds of workloads in virtual machine. In our experiments, we ran 8 VMs with Netperf (I/O intensive) , 8 VMs with Memcached (memory intensive) and 8 VMs with YCSB (data intensive) at same time. After running for a period of time (40s), we turned off the 8 VMs with YCSB, then 20s later (in time 60s) we rebooted these 8 VMs again and continue run for 40s. In this case, we recorded the network performance changes during the 100s. Figure 6 shows the network performance change. We found a performance degradation of 26.1% on VMs running Memcached and 25.3% on VMs running Netperf. When we reboot the VMs, the previous mapping relationship has been disturbed by the system's load balance mechanism, the VMM did not realize this change and result in the performance degradation.

Summary: the non-uniform assess degrade performance significantly in today's multicore system with high speed networking. Virtualization also poses a great challenge for optimizing the management of all kinds resources. These motivate our work and in the next section, we further explore affinity among different kinds resources and study optimal performance management methods.

3. ANALYSIS OF AFFINITY AND OPTIMAL MODELING

In this section, we begin with the necessity of managing the affinity in non-uniform access architecture. We then introduce a set of metrics that can be used to classify the affinity in this architecture. Lastly, we discuss the optimal affinity management model.

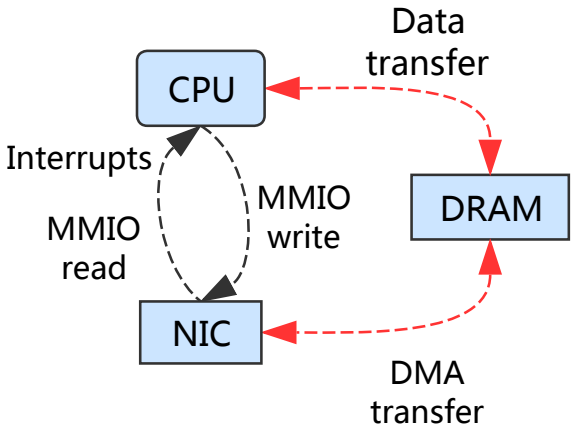


Figure 7: I/O data movement.

3.1 Necessity of optimal Affinity Modeling

Poor Affinity Affect Performance: As we experimented and analyzed in the section 2, non-uniform access architecture can not guarantee local access. Frequently remote access will cause the performance degradation in high performance networking environment. Non-transparent management in the virtualized system disturbs the distribution of all kinds of resources. All these factors finally result in poor affinity among the VM vCPU, memory and network adapter. Consequently, the inaccurate affinity affects the networking performance and results in the inefficiency of data center servers. Therefore, optimal affinity management model in the non-uniform access architecture is necessary.

Previous Affinity Model's Inaccuracy: Historically, the previous affinity model in the non-uniform access architecture mainly focus on optimizing the non-uniform memory access. Undoubtedly, these affinity models are no longer suitable for our non-uniform I/O access affinity modeling. Furthermore, previous affinity models are empirical that they just characterize the penalty of non-uniform access by using the hop distance. This distance is typically recorded as a table in the BIOS Advanced Configuration and Power Interface (ACPI) to measure the distance between NUMA nodes. However, the hop distance lack details in some cases and are insufficient for affinity modeling in asymmetric access architecture.

3.2 Affinity Classes and Metrics

We now discuss different affinity classes in the non-uniform access architecture. To accurately determine the affinity among CPU, memory and network adapter, we adopt more detailed metrics to describe the access behavior in the virtualized systems.

3.2.1 vCPU, Memory and NIC Interaction

We first address the main data movement in the non-uniform access . DMA technique is a dominant part that decouples processor involvement during data transfers between NIC and the Memory. As shown in the figure 6, when NIC receive and send data packets, data moves between NIC buffer and memory with DMA tunnel without many I/O request interrupts. The details of DMA technique are beyond the scope of this paper. Within virtualization, the data in NIC buffer uses IOMMU as address translator to access the physical address of VMâĂŹs memory. vCPU interacts with NIC mainly caused by interrupts. Wen the NIC buffer and the memory process data transmission, it generates I/O interrupts to interact with vCPU. vCPU directly accesses memory to read and write data locally or remotely.

3.2.2 Affinity between vCPU and Memory

vCPU to memory affinity refers to the traditional NUMA affinity between VM vCPU thread and its memory. Based on the previous work [], we standing in the point of view of the vCPU and expect the local memory access as much as possible. The potential for NUMA affinity of vCPU is proportional to the amount of memory access to each NUMA node. That is, if a vCPU accesses to the memory of one NUMA node frequently than others, it has high affinity with this node. The NUMA affinity for vCPU can be calculated with equation 1, where A^m is a vector that stores the affinity values between VM vCPU and each node, NC is a vector containing the number of memory access from VM vCPU to each node, and N is the number of nodes.

$$A_i^m = NC_i/(\sum_{j=1}^{N} NC_j) \quad (1)$$

This affinity value is minimal when vCPU threads do not accesses the memory in the node, in this case, the affinity equals to 0. The affinity achieves maximum when all the memory access to one NUMA node only, in this caes, the affinity is 1.

3.2.3 Affinity between NIC buffer and Memory

The NIC to memory affinity means the affinity between NIC ring buffer and VM's memory buffer. When a NIC receives incoming packets, it copies the data packets into the memory buffers using DMA. Obviously, if the buffer is allocated in the memory attached to the node which near to NIC, the access latency of NIC to memory is lowest. Otherwise, NIC would access memory buffer of the remote node by excess QPI links. In this caes, the QPI bandwidth is consumed quickly. We define the affinity between NIC buffer and memory by using the equation 3, where VMD is the VM memory page distribution table. For example, VMD_0 is memory of VM distribute on node 0 (which is near to the NIC), and $\sum_{j=1}^{N} VMD_j$ is the total number of VM memory.

$$A_i^n = VMD_i/(\sum_{j=1}^{N} VMD_j) \quad (2)$$

This affinity achieve best($A^n = 1$) when all the VM memory are allocated on the node near to the NIC, the minimal value of A^n is 0 when the VM do not allocate any memory on the node.

3.2.4 Affinity between vCPU and NIC

As we describe early, NIC interact with CPU mainly by interrupts and will not cause much interconnection consumption. Pervious work practically optimizing the IRQ (Interrupt request) affinity [] to achieving high performance, this affinity is defined as the set of processors that can service that interrupt [], it is not traditional distance affinity. It is better to distribute interrupt requests between all the available cores to obtain best IRQ affinity. In this work, we also optimize the IRQ affinity by tuning the network adapter.

3.3 Optimal Affinity Modeling

After addressing and measuring affinity among CPU, memory and NIC, considering complexity in the virtualized environment, we now discuss the optimal affinity model.

Manage memory and I/O affinity simultaneously: Current virtual machine run all kinds applications at same time to make use of powerful hardware resources. Data intensive application with frequently I/O access and computing intensive application with large amount of memory access. As we experimented in the section 2.2, simultaneous non-uniform memory and I/O access affect performance significantly. So our model should simultaneously optimize non-uniform memory and I/O access to achieve high performance.

Measure affinity more accurately: As we analyzed in the section 3.2.2, we should used more precise measurement of affinity instead of the hop distance. A^m measure the affinity between vCPU and memory, A^n measure the affinity between NIC buffer and memory. These affinities contain more underlying information about non-uniform memory and I/O access behaviour. Moreover, A^m and A^n can be dynamically updated and reflect the runtime changes of virtual machines.

Optimize affinity more dynamically: Since the virtualization layer is unawareness to the management of non-uniform access, our optimal affinity management must be dynamic and periodic. Dynamic resources migration is a efficient way to optimizing affinity in the runtime systems. In the consideration of system load and migration overhead, our model should apply a more comprehensive and low overhead migration mechanism.

Summary: In this section, we analyzed the affinity in the non-uniform access architecture in details and we also applied accurate metrics for these affinities. Lastly, we talked about that what is the optimal affinity model. In the next section, we provide a comprehensive design of our optimal management framework.

4. DESIGN

In this section, we describe the detail design of our *vNUIOA*, a runtime system that optimizing the overheads of non-uniform access in the virtualized environ-

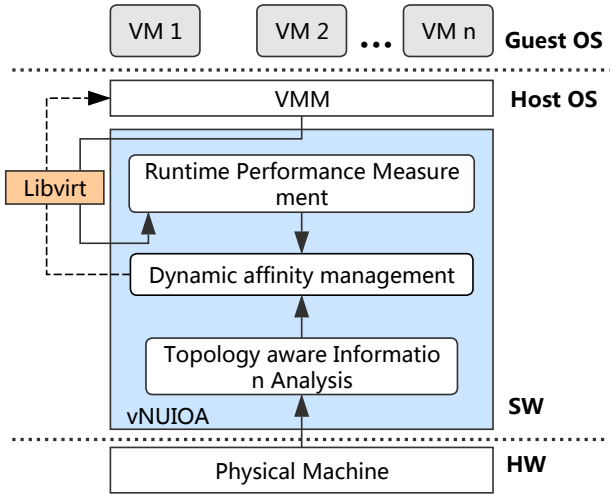


Figure 8: Architecture of vNUIOA .

ment. We enabling three parts for incorporating NUMA and NUIOA overheads awareness with hypervisor: namely, runtime performance measurement, topology aware information analyses and affinity driven dynamic virtual machine management.

4.1 Overview

An overview of *vNUIOA* is shown in Figure 8. It consists of the following three parts: the ***Runtime Performance Measurement*** module collects and stores the information of guest os since the system booted, then the ***Topology aware Information Analyses*** module analyzes and processes these information based on the topology of host machine. According to the virtual machine performance measurement and host machine analysis results, the ***Affinity driven Dynamic Virtual Machine Management*** module calculates the optimal placement with different affinity combination, dynamically migrates memory and maps vCPUs with the consciousness of system's loads. The details of each module are described as follows.

4.2 Runtime Performance Measurement

Runtime information measurement mainly collects the information we used in the calculation of affinity model. To periodically collect the performance information of virtual machines and host server, it is imperative to analyze memory and I/O access activities in the virtualized system. To end this, we use the Linux performance monitoring tool perfmon [] to detect memory and I/O access activities, then we store these information in the VMinfo table for each VM. Each table entry contains information about the accumulated number of VM access to each node vector(NC_n), VM memory pages distribution vector(VMD_n), and number of VM I/O requests(VIOR). The details of these information listed in the table 2:

4.3 Hardware Topology Analysis

Per VM statistics	
NC_n	the number of memory access from VM vCPU to each NUMA node n.
VMD_n	the distribution of VM memory pages on node n
VIOR	the number of I/O requests per second of VM .
Physical machine statistics	
L_i	CPU load of core i ($L_i = \frac{CPUtime_i}{Elapsedtime_i}$).
D_{mn}	Average access latency between NUMA node m and n.

Table 2: Definitions of collected information.

Due to the non-transparent management of Virtual machine monitor, an accurate understanding of underlying host machine topology is needed. We mainly analyse underlying topology the and runtime load per node by using the physical machine statistics (list in table 2).

Analyse the underlying topology: We first detect the which node(s) the NIC directly attached to, this information can be obtained by the Portable Hardware Locality (hwloc)[] software package which available in the Linux system. After locate the node(s) near to NIC, then we use the average access latency D_{mn} to measure connectivity between each node. From this analysis, we can understand the holistic topology of host machine and it is benefit for our optimization in the next section.

Analyse runtime load per node: Runtime CPU load is also important for performance of physical server. As we experimented in the section 2, physical machine system load balance mechanism will disturb the distribution of hardware resources. So in our design, we must analyse the real time loads based on the topology of host machine. we mainly consider average CPU load of each NUMA node,the per-node CPU load L_n can be calculated in the equation 4: Where M is the number of cores per NUMA node. We test that when the system load per node exceed the 80%, the system performance begins to decrease. Based on this experimental value, we define the threshold of L_n is 80%, if the system load per node up to 80%, it is regarded as high load.

$$L_n = (\sum_{i=1}^{M} L_i)/M \tag{3}$$

4.4 NUIOA and NUMA aware VM management

Because of the affinity unawareness management mechanism in the virtualized environment affects performance, a dynamic affinity management framework is need to keep runtime systems efficient. Dynamic management always with resources (memory, vCPU) migration, there are two kinds of migration methods:*push migration and pull migration*. In push migration, scheduler periodically checks the load of each processor and migrate threads from the processor with high load to processors with low load if needed. Pull migration refers to when a processor becomes idle and tries to steal threads

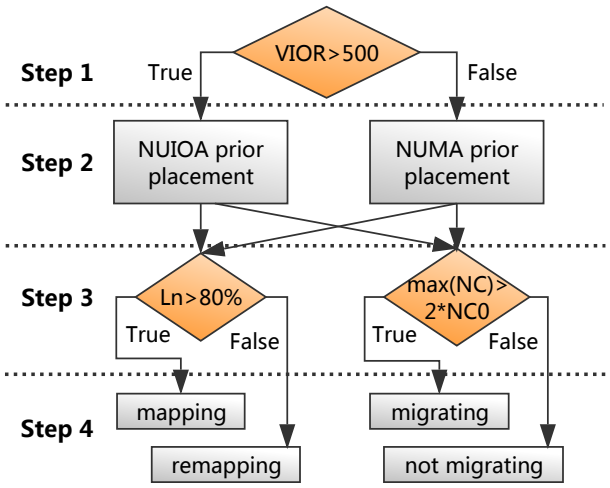


Figure 9: Flowchart of NUIOA and NUMA aware VM management.

form busy one. Our migration methods are closely to the push migration. The following steps show the mechanism of our affinity driven dynamic virtual machine management method.

Step 1: We first identify which type a virtual machine is, because for different type VM, we give different placement combination to achieve best performance. We classify the VMs into I/O intensive and I/O non-intensive by looking the VIOR vector(see the Table 2), the I/O request threshold is to be determined experimentally. We found that the number of I/O request for VM with running I/O non-intensive task is less than 100 per second and for I/O intensive task the number is more than 1000 per second, so we choose a threshold value of 500 times per second in consideration of practical application attributes.

Step 2: After each VM has been placed by the VMM default scheduler in the initial phase, we consider the our optimal placement combination for each VM.

NUIOA prior placement: According to the step 1, I/O intensive VMs always have frequent I/O operation, the non-uniform I/O access affects performance more. Our method is to migrate these VM memory buffer to node(s) near to NIC as much as possible. So the affinity A^n is calculated in the first place by using the equation 1, after place the memory buffer according to the best A^n, we then calculate optimal affinity A^m to keep memory local access by using equation 2, we denote this optimal placement priority as $<A^n,A^m>$.

NUMA prior placement: I/O non intensive such as CPU intensive VMs always read and write data with memory, in this case, we first map VM vCPU to node(s) which away from the NIC, then we calculate the best affinity A^m and migrate the VM memory pages to the local node according to the calculating results. Finally, we calculate the A^n to get the optimal memory buffer placement, this combination can be denoted as $<A^m,A^n>$.

Step 3: After calculating optimal VM placement in the step 2, we manage resources affinity dynamically and periodically (per second) via the vCPU mapping and memory migration. The vCPU mapping negotiates with the system load per node which we analyzed in the section 4.2. If the average CPU load per node is higher than 80%, we should select another node for vCPU mapping to avoid contention. We remap the VM vCPU from the original node n to node k instead of the node p is beneficial only if the average access latency to node k (D_{nk}) is less than average access latency to node p (D_{np}). remember L_k and L_p both less than 80%. We only migrate memory when maximum number of remote node memory access is double than the number of local memory access ($max(NC_n) > 2 * NC_0$), because frequent memory migration also bring high overheads. We test several value and found that when select double than the number of local access, we can achieve max benefits.

Step 4:After finishing the memory migration and vCPU remapping, we update the VM Info table for next turns of VM management.

5. IMPLEMENTATION

We implemented the vNUIOA prototype in the host operating system with KVM virtualized platform. The holistic system is implemented as individual daemous, we now describe the implementation details of our system.

We used the KVM 2.0.0 as virtual machine monitor with the Intel VT-d and VT-x technology enable. Each VM run the Ubuntu 14.04LTS system and configured with one vCPU and 1GB memory. In order to achieve high performance I/O virtualization, we enabled Single Root I/O virtualization (SR-IOV)[] technology and assigned a virtual function to each VM. SR-IOV [] proposes a set a hardware enhancements for PCIe device and enable to create multiple "lightweight" PCIe functions,known as virtual Functions (VFs) that still contain the major device resources. In this work, we mainly consider SR-IOV because With the hardware enhancement, SRIOV removes the major VMM intervention and achieves I/O virtualization without losing performance.

5.1 Fast and accurate performance measurement

Using page fault for memory access sampling: Our *vNUIOA* first should detect the memory page access quickly and accurately and with low overhead. Previous work [] has investigated that the OS has a methods to measure if the pages has been accessed by the page fault, While the page is not set as present in the page table, the OS is noticed about the faulting virtual address, as well as the thread ID that caused the page fault. By tracking page faults, it possible to detect which vCPU threads are accessing the memory page. We use Memory Page Faults with vCPU theads id in the Linux performance monitor tool *perfmon* to sample the access number to each node of the vCPU thread.

Acquire VM memory allocation via numa_maps:

The */proc/<pid>/numa_maps* file displays information about a process's NUMA memory allocation. Since each virtual machine is regarded as a process in the Linux system, we can acquire the information about the pages allocated in the NUMA node according to the VM <pid>. By looking at the N<node>=<nr_pages> line, we can obtain the number of pages allocated on node N in the <nr_pages> and we read this value in the *VMD* vector.

Counting I/O requests per second: Every time when a VM handles a packet, the SR-IOV vf driver invokes *ixgbevf_clean_tx_irq* or *ixgbevf_clean_rx_irq* function for every second. We count the execution number of these two functions for the number of I/O request per second and store this information in the *VIOR* data structure.

5.2 Efficient management of memory and vCPU

Migrating pages via libnuma system call: After determining memory migration strategies, we use the *numa_migrate_pages()* system call to migrate all pages of the specified process from one node to another node. This system call is supported in the **numactl** package.

Binding vCPU via libvirt API: **libnuma** is software API for managing virtual machines. It support various domain lifecycle operations such as VM start, stop and migrate and we use the *sched_setaffinity()* function to set CPU binding policies for VM.

6. EVALUATION

In this section we present experimental evaluation of the proposed vNUIOA system, we compare the performance of vNUIOA with KVM's default scheduler and a hand optimized binding strategy. Then we study the stability of our vNUIOA system. Finally, we characterize vNUIOA runtime overhead.

6.1 Experimental Environment

All experiments are conducted in a 4-sockets machine with Mellanox ConnectX-3 40 GigE adapter placed on the PCIe Gen3 x16 slots which directly connect to the socket 0. Each socket contains a Intel Xeon Ivy Bridge architecture processor and 32GB DDR3 memory bank. Each processor with 8 cores runs at 2.3GHz and max turbo frequency can up to 2.7GHz. Each core has 32KB L1 data cache and 32KB L1 instruction cache, 256KB L2 cache and 16MB last level cache (LLC). Each socket equips with 2 ways QPI with the communication speed 7.2GT/s. In our experiments, we disable the Intel Hyper-threading []. Therefor, we have total 32 cores and 128GB memory, we also configure the storage with 300G disk. Each VM with a vCPU and 1GB memory. The experiment configuration details are listed in the table 3.

We select a set of network and memory performance testing benchmarks. The Netperf [] benchmark provides network bandwidth testing between two host on a network. Memcached [] is an in-memory key-value store for small chunks of arbitrary data to improve the performance of cloud applications. Yahoo! Cloud Serving Benchmark (YCSB) [] is often used to compare relative performance of database read,write and update management systems.

Item	Configuration
CPU	Intel Xeon 4-sockets processor each with 8 cores (2.3GHz)
Memory	128G RAM DDR3 4 sockets, each with 32GB
QPI	7.2 GT/s, 2 links
Network Adapter	Mellanox ConnectX-3 Dual-Port 40 Gigabit Ethernet adapter
Hypervisor	KVM 2.0.0

Table 3: Configuration details of our test server.

We evaluate three scheduling strategies: the KVM default scheduler, hand optimized and our *vNUIOA*. The default scheduler in the KVM system is numad [], numad is an automatic NUMA affinity management daemon in the Linux system, it can dynamically scheduling the NUMA system resource allocation by monitoring system topology and resource usage. The hand optimized strategy is simply binding VM vCPU and memory together to the nodes near to the network adapter. *vNUIOA* also dynamically manages the system resources allocation and migration based on the asymmetric memory and I/O access architecture.

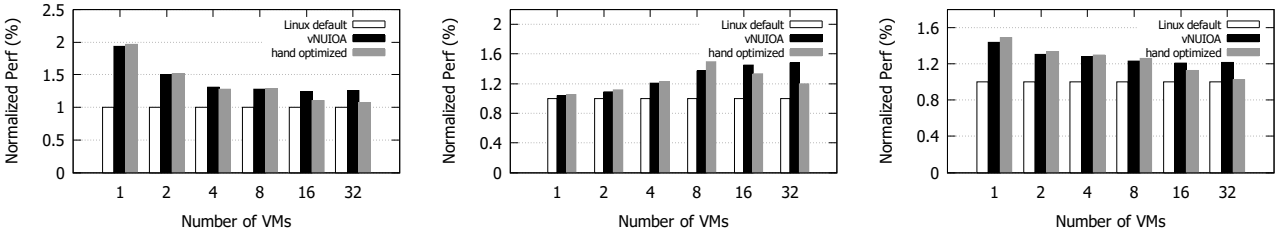
6.2 Improvement on workloads Performance

Figure 9 and 10 shows the performance comparison of different benchmarks under three different management strategies. For each benchmark we run ten times under three strategies, we calculate the average performance of ten runs and it normalized to the KVM default scheduler numad.

6.2.1 Single workloads scenario

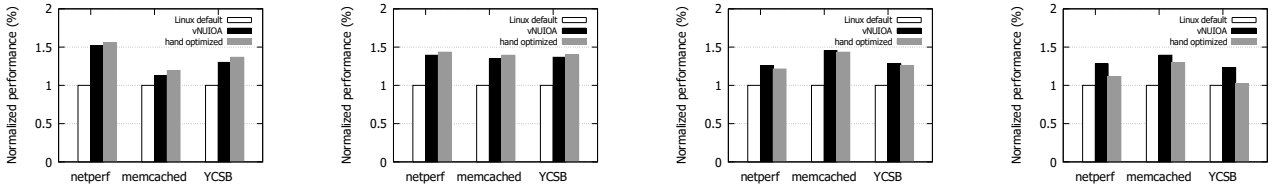
For network throughput improvement (Figure 9(a)), *vNUIOA* outperformed numad by at least 23.3% (7 VMs) and by as much as 93.9% (1 VM). Since our *vNUIOA* prior map VM vCPU and memory to the node near NIC. With the number of VM increases, bandwidth of NIC has been occupied and the improvement is steady (23%). As for hand optimized strategy, *vNUIOA* performed slightly lower due to the overheads. But with the number of VM increases, binding too much VMs to one node incurs resources contention and affects the performance. When the VM number up to 7 and 8, the performance of hand optimized strategy decreases respectively 12.8% and 16.3%.

Figure 9 (b) shows the improvement on memory intensive benchmark. From the figure, the performance of *vNUIOA* close to the hand optimized strategy with no more than 3.8% performance reduction when the number of VM up to 6. However, with the the number of VM increases, hand optimized strategy brings more resources contention and result in performance degradation (7 VMs drop 15% and 8 VMs drop 25.7%). This proves that when the number of VMs increases, the



(a) Netperf (b) Memcached (c) YCSB

Figure 10: Evaluation results of Single workloads



(a) 3VM (b) 6VM (c) 9VM (d) 12VM

Figure 11: Evaluation results of mixed workloads

statically hand optimized strategy is inefficiency. Comparing to the numad strategy, *vNUIOA* has an average 26.4% and up to 47.9% improvement.

For both data and computing intensive benchmark (YCSB), our *vNUIOA* performs better to the numad when the VM number is small. With the number of VM increases, *vNUIOA* can improve performance closed to 20%. Hand optimized strategy achieves high performance when there is smaller VMs. Large number of VMs under the hand optimized strategy also affects perforamnce (8 VMs 10% degradation relative to numad), we can predict that hand optimized strategy will hurts performance more when the number of VMs continuously increases.

6.2.2 *Mixed workloads scenario*

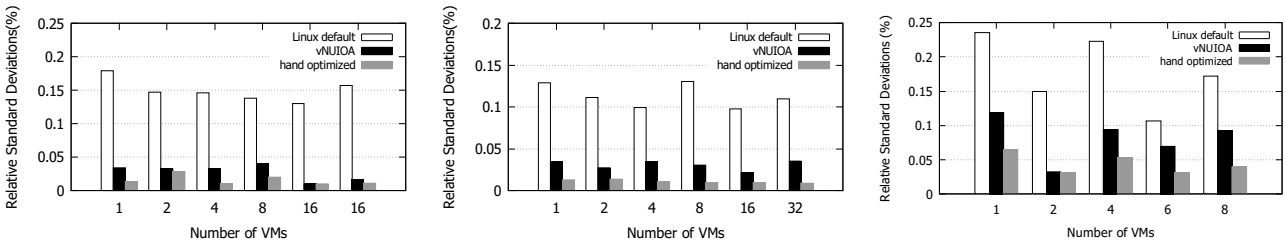
In this scenario, we simultaneously ran three different types VM, one with netperf, one with memcached and another with YCSB. We gradually add the number of each type VM up to 4 (total with 12 VMs) to evaluate our *vNUIOA*. Figure 10 shows the result, when the number of VM running each type workload is 1, our *vNUIOA* achieve an average 45.4% improvement relative to the numad. For each type workload, with the number of VM increases, our vNUIOA can improve performance at least 20%. For the hand optimized strategy, when the number of VM is small, it has the best performance. But with the VMs number increases, this strategy will affects performance on the contrary (4 VMs with YCSB).

6.3 Reduction on performance Variation

In order to show the stability of our *vNUIOA*, we compare vNUIOA with other two scheduling strategies in terms of performance variations. We use the *relative standard deviations (RSD)* of ten individual runs to measure the variability of different strategies. The smaller the RSD value, the more steady and predictable workload performance. As shown in the figure 11, for hand optimized strategy, because it has fixed memory mappings and vCPU bindings, it always achieve small RSD values in all kind workloads (average RSD=1.2%). The KVM default scheduler numad, due to the influence of topology unawareness management of KVM system, causes the RSD values considerable. For network throughput testing workload, the average RSD values of numad scheduler is as large as 14.4%. For memory intensive workload, numad had a smaller variations (11.3%), but it also larger than our *vNUIOA* (2.5%). The average variations of our *vNUIOA* is closed to the hand optimized strategy. All above data can prove that vNUIOA performs stability in all kind workloads.

6.4 Overhead

Since *vNUIOA* operates during the execution of virtual machine monitor, it causes overheads. The main runtime overhead of *vNUIOA* can be attributed to the following reasons: collecting and storing the performance information, calculation of the optimal affinity function and migration of pages and threads between nodes. For storage overhead, in our test bed (4 nodes, 32 cores), the *NC* vector is 4 bytes and *VMD* vector is also 4 bytes, *VIOR* is 1 byte, so for one VM, we just needs 9 bytes. Storing the CPU load of cores L_i need 32 bytes and *DL* table needs 4*4=16 bytes. These storage overhead is negligible. For runtime overhead, we measure the execution time of each workload and the results shown in the figure 12.



(a) Netperf (b) Memcached (c) YCSB

Figure 12: Reduction on workloads performance variation

7. RELATED WORK

In this section, we explain how our work relates to different works on multicore systems in the virtualized environment. We first review works aimed at optimizing asymmetric I/O access overhead both in academic and industrial, then we contrast our vNUIOA with previous affinity driven systems.

Non-uniform Access Architecture: Previous research mainly addressed the non-uniform memory access in the multicore systems[]. With the speed of network adapter getting higher and higher, existing work has been realized performance degradation due to non-uniform I/O access in variety research area. For example, Li *et al.*[] characterize the I/O bandwidth performance in NUMA architecture for data intensive applications. Rodiger *et al.*[] consider the impact of Non-Uniform I/O access architecture in the high speed database query processing over high speed networks. In the virtualized environment, Xen[] and VMware[] are two typical systems providing NUIOA support. Xen community has already realized the benefits of running VMs on the nodes which I/O devices directly attached. VWware has implemented a NUMA aware I/O scheduler in VMware vSphere 6.0. However, not maintaining one resource locality can achieve high performance. Due to the heterogeneous of virtual machines and fairness management of VMM, it is complex to establish a uniform relationship model between all kinds resources for different types VM. Several studies [] use the performance monitor unit (PMU) to monitor online resources useage for performance modeling, but for some cases the performance index which PMU provide is not enough comprehensive. Different form these above solutions, our vNUIOA system both takes non-uniform memory and I/O access into consideration.

Affinity Modeling: Affinity modeling is widely used method in performance optimization under non-uniform access architecture. Diener *et al.*[] managed both thread and data affinity by a automatic kernel-level method in NUMA system, and Majo *et al.*[] aimed to achieve a optimal performance by making a trade off between memory affinity and cache contention. However, these affinity models only measure the relationship between two factors, as the system get virtualized, these models are not suitable for our affinity modeling among three factors(vCPU, memory, NIC). Modern OS such Linux provides *libnuma*[] tool for program level affinity modeling, but *libnuma* relies on the hop distance which is statically recorded in the BIOS ACPI table. As we discussed in the early section, hop distance sometimes may be incorrect or over simplified and finally brings about imprecise model.

Runtime Dynamic optimizations: Most of previous performance optimizations on non-uniform access architecture are achieved by tuning manually. Dell has proposes a series fine tuning based on NUMA I/O locality for some workload types on its PowerEdge 12th generation servers []. Mellanox [] also supports performance tuning for its infiniband [] network adapters, providing tuning methods both for Intel and AMD platform. However, in a runtime virtualized system, programs are running dynamically and due to dynamic management of virtualization layer, the performance tuning is ineffective sometimes. Our approach not only obtains performance index online, but also takes the runtime system load into consideration. Based on these, our approach can ensure efficiency of optimization all the time.

8. CONCLUSION AND FUTURE WORK

We showed that the asymmetric access drastically impacts network performance of multicore systems. The non-transparent management of virtualization impose another significant challenges to achieve optimal workloads performance. To address these issues, we first analyzed the I/O and memory access behaviors in virtualized environment. Then we provided a affinity model which simultaneously takes CPU, memory and NIC into consideration. Finally, we introduced *vNUIOA*, which is a framework that dynamically manages affinity based on the hardware topology. We implemented *vNUIOA* in KVM virtualized platform. Experiments with different type workloads showed that *vNUIOA* was able to improve performance substantially. *vNUIOA* improved workloads performance by up to 93.9% (32.9 on average) compare with the Linux default scheduler *numad*. *vNUIOA* also provided a steady performance with no more than 5% variations.

For the future, we intend to expand *vNUIOA* to other virtualized platform like Xen. Another extension of our future work is to design a more efficient heuristic algorithm to calculate the mapping combinations with-

in shortest execution time. Thus our *vNUIOA* system will be market-oriented.

9. ACKNOWLEDGEMENTS

This research was supported by the National Science Foundation under grants

10. REFERENCES

[1] F. Lastname1 and F. Lastname2, "A very nice paper to cite," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[2] F. Lastname1, F. Lastname2, and F. Lastname3, "Another very nice paper to cite," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[3] F. Lastname1, F. Lastname2, F. Lastname3, F. Lastname4, and F. Lastname5, "Yet another very nice paper to cite, with many author names all spelled out," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011.