ResourceExchange: Latency-Aware Scheduling in Virtualized Environments with High Performance Fabrics

Adit Ranadive, Ada Gavrilovska, Karsten Schwan
Center for Experimental Research in Computer Systems (CERCS)
Georgia Institute of Technology
Atlanta, Georgia USA
{adit262, ada, schwan}@cc.gatech.edu

***Abstract*—Virtualized infrastructures have seen strong acceptance in data center systems and applications, but have not yet seen adoptance for latency-sensitive codes which require I/O to arrive predictability, or response times to be generated within certain timeliness guarantees. Examples of such applications include certain classes of parallel HPC codes, server systems performing phonecall or multimedia delivery, or financial services in electronic trading platforms, like ICE and CME.**

In this paper, we argue that the use of high-performance, VMM-bypass capable devices can help create the virtualized infrastructures needed for the latency-sensitive applications listed above. However, to enable consolidation, problems to be solved go beyond efficient I/O virtualization, and include dealing with the shared use of I/O and compute resource, in ways that minimize or eliminate interference. Toward this end, we describe *ResEx* – a resource management approach for virtualized RDMA-based platforms which incorporates concepts from supply-demand theory and congestion pricing to dynamically control the allocation of CPU and I/O resources of guest VMs. ResEx and its mechanisms and abstractions allow multiple 'pricing policies' to be deployed on these types of virtualized platforms, including such which reduce interference and enhance isolation by identifying and taxing VMs responsible for resource congestion. While the main ideas behind ResEx are more general, the design presented in this paper is specific for InfiniBand RDMA-based virtualized platforms due to the use of asynchronous monitoring needed to determine the VMs' I/O usage, and the methods to establish the trading rate for the underlying CPU and I/O resources. The latter is particularly necessary since the hypervisor's only mechanism to control I/O usage is by making appropriate adjustments in the VM's CPU resources.

The experimental evaluation of our solution uses InfiniBand platforms virtualized with the open source Xen hypervisor, and an RDMA-based latency-sensitive benchmark, BenchEx, based on a model of a financial trading platform. The results demonstrate the utility of the ResEx approach in making RDMA-based virtualized platforms more manageable and better suited for hosting even latency-sensitive workloads. ResEx can reduce the latency interference by as much as 30% in some cases as shown.

I. Introduction

Virtualized infrastructures have seen strong acceptance in data center systems and applications, but have not yet seen adoptance for HPC codes which require I/O to arrive within predictably and consistently short durations (i.e., no 'noise'), in exchanges like ICE, CME, and NYSE which need trades to complete in a certain amount of time (deadlines), also in server systems performing phone call switching or multimedia delivery, which require soft deadlines to be met. At the same time, however, there is a need for virtualization in such environments, as demonstrated by recent work in which soft deadline capabilities are added to the Xen hypervisor in order to support VOIP call switching server applications [11]. In fact, there are even greater opportunities for virtualization for exchanges, since their average machine utilization can be less than 10% under normal load conditions, due to conservative provisioning in order to meet peak demands and high availability requirements. There are also opportunities with HPC codes, for which data centers routinely report issues with under-utilization despite intensive user of batch job schedulers, and these opportunities will grow as leadership machines move to the exascale, due to the difficulties many programs will have to efficiently and concurrently use resources at that scale.

In this paper, we develop mechanisms to better enable hosting and collocation of latency-sensitive applications, such as those listed above, on virtualized platforms. We are specifically targeting high-performance RDMA-enabled interconnects, for two reasons. First, these devices have features such as higher bandwidths ($\geq$10Gbps), as with InfiniBand [8] and 10GigE [13] network fabrics, support for remote DMA, protocol offload, and kernel bypass, and, as a result, they provide the low-latency high-bandwidth requirements needed by the aforementioned types of applications. Second, the same set of features also enables efficient near-native I/O virtualization of these devices [18], which makes them more suitable candidates for virtualized platforms for latency-sensitive applications.

However, to enable consolidation for latency-critical applications, problems to be solved go beyond efficient I/O virtualization, to also include dealing with the shared use of I/O, memory, and compute resources, in ways that minimize or eliminate interference. Newer generation InfiniBand cards allow controls such as setting a limit on bandwidth for different traffic flows and giving priority to certain traffic flows over others, thereby controlling latency for that flow. Also, what is required is for the hypervisor scheduler to be enhanced to make sure that it can take into consideration the latency

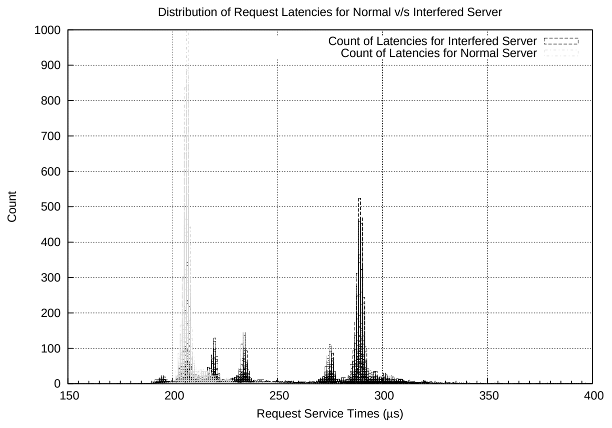

Fig. 1. Distribution of latencies of a normal server versus an interfered server with additional load.

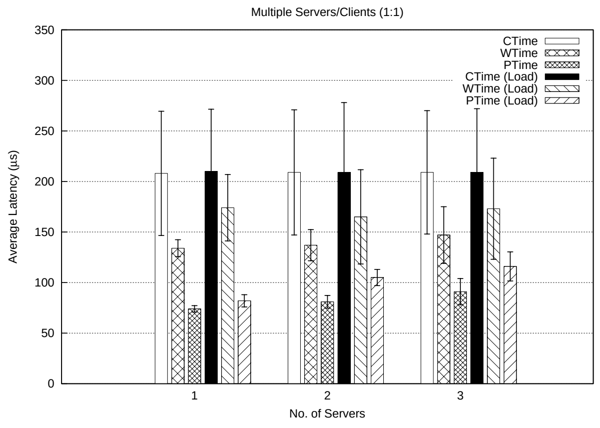

Fig. 2. Change in server latency for multiple servers with interfering load.

requirements of the VMs, the loads they pose on the shared network infrastructure, and the levels of interference and noise they cause due to their I/O access patterns. This is particularly challenging in virtualization solutions for RDMA devices such as InfiniBand HCAs, since due to the VMM/OS-bypass capabilities provided by these devices, the hypervisor loses control in monitoring and managing device-VM interactions. Therefore, it becomes harder for the hypervisor to maintain data rate or latency service levels to its guest VMs.

To address these issues, we develop a hypervisor-level solution which dynamically adjusts the compute and I/O resources made available to each VM, so as to provide better isolation – i.e., control noise – and improve performance for latency-critical applications, while still allowing consolidation and improved aggregate resource usage, without requiring worst-case-based reservations. Our approach, termed *Resource Exchange* or *ResEx*, leverages (i) IBMon – a tool developed in our prior work [19] that enables dynamic monitoring of the I/O usage for virtualized InfiniBand devices, and (ii) congestion pricing models and supply-demand microeconomic concepts to provide more flexible and fine-grained resource allocations.

Specific contributions of our research include (1) the design and implementation of the *Resource Exchange* approach, and (2) its resource trading 'currency' abstraction – *Resos*, and (3) mechanisms for dynamic monitoring of VMs' resource usage and the interference, or the 'congestion' they cause, so as to be able to (4) support range of resource pricing policies derived from microeconomic theory on supply-demand and congestion pricing. (5) Using BenchEx – a latency-sensitive benchmark modeled after a financial trading exchange, (6) we implement and experimentally evaluate two such policies, and demonstrate the feasibility of the ResEx approach to make RDMA-based virtualized platforms more manageable and better suited for hosting even latency-sensitive workloads.

The remainder of the paper is organized as follows. Section II uses experimental data to show the effect of collocation on latency-sensitive applications. In Section III we give a brief description of the different software and technologies we use, and the economics pricing theory we rely on. Section IV describes the latency-sensitive benchmark that we have developed and use throughout the paper. In Sections V and VI we describe in detail the design goals and implementation for ResEx. Section VII provides experimental results demonstrating the benefits of ResEx. Finally, we give a brief overview of related work in VIII and conclude at the end.

II. Motivation

While high-end fabrics, such as the InfiniBand fabric considered in our research, exhibit high bandwidth and low latency, their shared usage for latency-sensitive workloads has been limited. When running a latency-sensitive workloads on shared network infrastructure it is expected to observe some level of perturbation in the achieve latency. Under some conditions, the variability may be within acceptable levels, and/or may be exhibited only by some of the collocated applications, and not by the most latency-sensitive ones. In other cases, however, this variation can exceed desired limits, especially when the degree of interference with other applications is high.

Our goal in this section is to demonstrate that, in spite of the superior performance properties of high-end fabrics like InfiniBand, their shared usage by latency-sensitive applications may result in prohibitive levels of variability. For this, we explain the changes experienced by a low latency workload when it is being run with (Interfered) and without (Normal) another interfering workload. We use different configurations (i.e., different data sizes) of the BenchEx latency-sensitive benchmark developed by our group (explained in Section IV) as both a low latency workload and second as an interference generator, each run in a separate VM. Both benchmark instances, i.e., both VMs, are assigned to separate CPUs, therefore there is no response-time variability due to CPU scheduling.

Figure 1 shows the frequency distribution of the low latency workload when it is run with and without the interference load. In the Normal case the latencies are highly stable at around 209 μs. But when it is run alongside the interfering load the latencies are distributed across the interval. This shows that not only the average increases but the variation/jitter as well even for RDMA-based interconnects. Note that for certain requests the service time is smaller than Normal server possibly due to no interference occuring for those requests.

Figure 2 shows the change in server latency as server and client count is increased, as well as when there is interfering load added to the system. There are three parts to the server latency - Compute Time (CTime), I/O Wait Time (WTime), Polling Time (PTime). Each of them is shown with the variation in error bars. Here, the number of servers refers to separate instances of the low-latency workload configuration of the benchmark. For each group of servers, we report average latency when these are the only collocated applications on the node, vs. when also run with a VM running an interference generator. Again, all VMs are allocated to their own CPUs. Since CTime is independent of I/O interference it remains fairly constant. From the figure we observe that both WTime and PTime start increasing with load since now a RDMA operation from the server takes more time to process due to the interfering load at the device level. We also observe, however, that when collocating only the VMs running the original application, the interference effects, and any latency degradation, are much less noticeable, demonstrating that it is feasible to allow latency-sensitive workloads to share the virtualized platform.

III. Background

ResEx relies on certain properties of the underlying fabric – InfiniBand – and hypervisor – Xen, in our case. These, as well as the ideas from economics and congestion pricing used in ResEx, are described next.

InfiniBand Memory Management. The communication model used in IB is based on the concept of queue pairs (Send and Receive Queues). A request to send/receive data is formatted into a Work Request (WR) which is posted to the queues. There are also Completion Queues (CQs) which store Completion Queue Entries (CQEs) containing information for completed requests. Whenever a descriptor is posted to the queue pair (QP), certain bits called 'Doorbells' are set in User Access Region (UARs) buffers. The UARs are 4KB I/O pages mapped into the process' address space. When a request is issued, the doorbell is "rung" in the process' UAR. The HCA will then pick up these requests and process them.

The HCA maintains a table called the Translation and Protection Table (TPT). This table contains the mappings from physical to virtual addresses of the buffers and queues described above. For InfiniBand, the buffers need to be registered with the HCA, and these registry entries are stored in the TPT. Also, these buffers must be kept pinned in memory during data transfer so that the HCA can DMA the data directly in and out of host memory. Each entry in the TPT is indexed with a key that is generated during the registration process.

Virtualized InfiniBand Driver Model. In Xen, device virtualization typically follows the 'split device driver' model [7]. For InfiniBand, the split-driver model is used slightly differently. The control path operations from the guest VM, like memory registration and queue pair creation, must all go through the backend driver in dom0. Fast path operations like polling the CQ and QP accesses are optimized using a VMM-bypass [12] technique. The data path is highly optimized since the HCA can write DMA data directly to the target buffer. As a result, IB platforms can be virtualized with negligible impact on the attainable latency and bandwidth.

Virtual Memory Introspection. With the Xen VMM, pages of one VM can be read by another VM simply by mapping the page into the target VM memory. This concept was first introduced in [4]. In Xen this is done through the XenControl library. Using the function *xc_map_foreign_range*, the target memory is mapped into the current application's virtual memory. We use this mechanism to map the physical pages which correspond to the IB buffers into the monitoring utility.

Xen Library Interfaces and Scheduler. We use the *XenStat* library to interact with the Xen hypervisor. This library allows us to get and set the CPU consumed by the VM. ResEx uses the information from IBMon (described below) to make an assessment of the amount of CPU to be provided to the VM depending on the amount of congestion detected. The amount of CPU to be provided is called a 'CPU cap'. During scheduling, the Xen hypervisor allows the VM to run only for a percentage of its time slice (10ms), depending on the value of the 'CPU cap'. Also, in general we assign a whole VCPU to a VM so as to minimize the effects of shared CPU resources as described in [20].

InfiniBand Monitoring - IBMon. We have developed a tool called IBMon [19] which uses the memory introspection technique described above to map VM IB memory (with some assistance from the dom0 device driver) to the IBMon application running inside dom0. IBMon then periodically monitors this memory region to extrapolate the different characteristics of the IB application running inside the VM. This is similar to any out-of-band monitoring for networks. The characteristics that IBMon detects are number of completed requests by HCA for an application, size of the buffer used by the application, QP number used by the application. Since this memory is updated by the HCA continuously we can also track the rate of change of the entries to know how many entries were written in a specific time interval.

Economics and Congestion Pricing. Many of ideas that we incorporate into ResEx have been motivated by principles of microeconomics like supply and demand [6]. By supply we mean the set of physical resources that are available to each VM to use in a given time period or 'epoch'. The demand is the amount of resources that each VM consumes in that 'epoch'. Microeconomics theory states that when the demand for a commodity goes up the price for that commodity increases and the converse is true as well. We use this idea as the main motivator for reducing congestion on shared resources by VMs.

This idea is more expressly presented in another concept of economics derived from supply and demand called *'congestion pricing'* [22]. By increasing the price of a resource we aim to reduce the demand for it thereby reducing the 'congestion' for that resource. Congestion arises since the demand or 'resource usage' exceeds the supply or 'resource availability'. Changing the price for this resource corresponds to charging the heavy users for causing the congestion. We describe this in more detail in Section V-C.

IV. BenchEx - RDMA Latency-Sensitive Benchmark

In order to evaluate the behavior of latency-sensitive workloads on virtualized RDMA-enabled fabrics, we develop an RDMA-based latency evaluation benchmark, which allows us to easily parameterize and experiment with this class of applications. The benchmark, termed BenchEx, is modeled after a commercial trading engine used by one of our industry collaborators (ICE – Intercontinental Exchange) [21]. The benchmark uses a server-client model similar to trading systems where multiple clients post transactions and request feeds from a trading server hosted by the Exchange, and it includes traces which model the I/O and processing workloads present in an exchange like ICE [21] today. Furthermore, the benchmark further allows us to easily configure it and change its I/O behavior – e.g., rates, sizes, etc. – or per-request processing times, thereby simulating applications with different performance and resource usage profiles.

In BenchEx, our clients send multiple transaction requests to the server using RDMA, each of which is time stamped by the client. The server maintains these requests in a queue and processes them in a first come first serve basis. The FCFS criterion is important to exchanges since each transaction may change the outcome of the next one and we implement the same criteria for our benchmark.

In addition to trace data, since we do not have the proprietary data and processing codes used in real exchanges, our server uses a financial processing algorithm library [1] for request processing to perform operations such as Black Scholes Options Pricing. The clients wait for the server results, timestamp the reply, and calculate the latency of the request by taking a difference between the two timestamps.

BenchEx also provides an online monitoring interface to an external agent, running inside each VM, through which it can continuously report the observed server-side latencies. The agent may then forward this information to the main ResEx module running in Dom0, as described in Section VI.

V. Design Principles of ResEx

In order to provide a better understanding of our implementation of ResEx, we next describe the methodologies leveraged in its design. First, we describe how we can track InfiniBand I/O usage of VMs and therefore the I/O interference between VMs. Second, we introduce a currency called Resos to charge VMs based on their usage of IB and CPU resources. We describe how we allocate and price the Resos for a resource. Finally, we outline the various goals of our resource pricing schemes and discuss alternatives on charging VMs.

A. Tracking I/O Usage and Interference for VMM-Bypass InfiniBand Devices

Since the VMs can directly talk to the devices it is almost impossible for the hypervisor to estimate I/O usage for such devices. We use our IBMon tool [20] to track the amount of IB resources that a VM is consuming. By using IBMon we can estimate the amount of data that a VM sent in a given interval of time. This amount of data corresponds to a number of packets that the HCA has to send on behalf of the VM. Each packet size is equal and is referred to as a MTU (Maximum Transmission Unit). By knowing the size of the MTU we can infer the number of packets that a VM sent. Therefore, we track the 'number of MTUs sent' (MTUSent) by a VM as a metric for I/O usage.

Each application within a VM uses a buffer size which may be different for each application. We call the ratio of an application's buffer size to another application's as the 'buffer ratio' (BR). We also use this metric for estimating the amount of I/O performed by one VM compared to another.

We define 'interference' as the positive change in any I/O latency or a negative change in device usage perceived by the VM or the application within the VM. First, a direct way to track interference is to get feedback from the application in terms of the latency it perceives from its requests. Second, an indirect way, would be to infer from the IB memory how many requests were completed by the HCA on the VM's behalf in a given time interval and then decide the percentage of bandwidth used by a VM. By knowing the percentage of bandwidth consumed we can estimate its impact on other VMs. Once we can estimate the impact of one VM on another's latency we aim to control it using CPU scheduling or capping the interfering VM's CPU. We discuss those ideas next.

B. Relationship between IB I/O Latency, Buffer Ratio and CPU Cap

In order to determine how to best control I/O latency interference across VMs, we perform several experiments to establish the dependencies between various parameters of the VM's I/O and CPU usage.

Figure 3 shows the change in I/O latency for a VM when another interfering VM is running. We run our BenchEx application within each VM. Each instance has a different application buffer size. We report latencies for the server side of the benchmark. The number in brackets corresponds to buffer size of the interfering VM. The number before the brackets is the 'buffer ratio' of the 'interfering VM' with respect to the 'reporting VM'. The buffer size of the reporting VM is set to 64KB. We compare latencies achieved by the reporting VM when the CPU cap of the interfering VM is set according to the buffer ratio. For example, the CPU cap for a 256KB VM is set to $100/4 = 25\%$. Since the latencies experienced by the reporting VM do not change between all the instances we can say that the CPU Cap has a direct relationship with the buffer ratio and I/O latencies experienced by a VM.

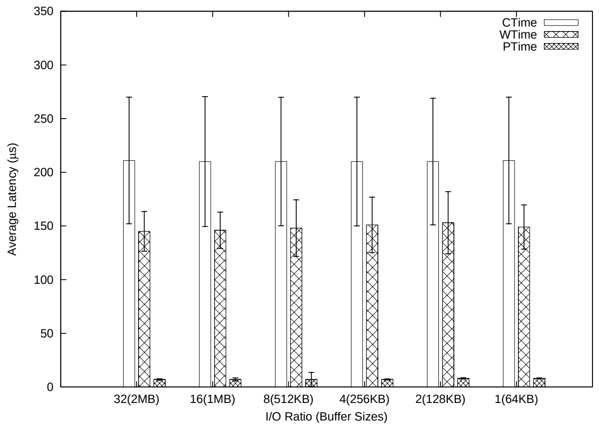

Fig. 3. Histogram of server latency with different I/O ratios between interfering and normal servers.

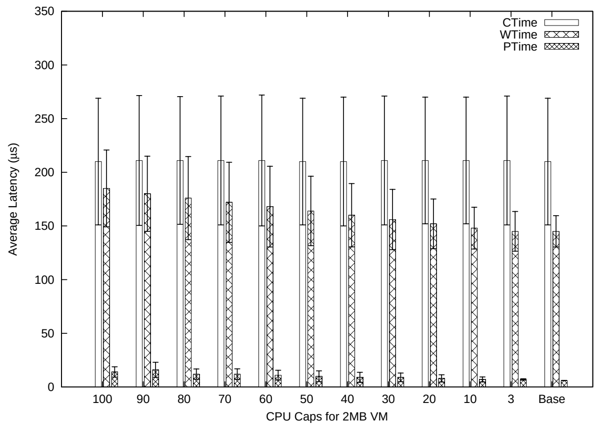

Fig. 4. Change in normal server latency as CPU cap is decreased for interfering VM.

In Figure 4 we set the buffer size of the interfering VM to 2MB and compare the latencies of the reporting VM when the CPU cap for the interfering VM is set accordingly. We see that by changing the CPU cap steadily the latencies experienced by the reporting VM decrease and when the CPU cap is equivalent to the buffer ratio-based value the latency experienced is equal to the base latency.

Therefore, by knowing the buffer sizes of the VMs and I/O latency experienced by the VM we can identify an interfering VM and assign a particular CPU cap in order to remove congestion. We use this as an indicator for our ResEx algorithm as described in Section VI in order to provide a consistent CPU allocation for the VMs. We base the consistent CPU allocation by unifying the resources used by a VM under a single entity which we describe next.

C. I/O Economics and Pricing

We introduce the concept of 'Resource Units' or *Resos* using which VMs 'buy' resources to use during their execution. Here, as 'buy' we refer to the ResEx ability to deduct Resos from the VM, based on its resource usage. Each Reso enables the VM to buy a certain amount of CPU and IB MTUs; sending more data will result in deduction of more Resos. The total number of Resos in a system is determined by the entire set of available physical resources in the system – the 'supply'. This set, in our case, comprises of the InfiniBand bandwidth link and CPU Cycles. We first determine the aggregate available resources, i.e., the corresponding aggregate Resos, and then distribute this equally among all collocated VMs. The Resos can also be distributed unequally, e.g., based on priority of the VMs. VMs can be charged for their resource usage differently, depending on the level of interference they cause, and based on the goals of the pricing algorithm (see the following section). Summarizingly, Resos is a mechanism to abstract different types of resources and their availability for each VM.

Resos are deducted at every interval for a VM based on its I/O and CPU usage, as described in Section V-A. A collection of intervals is called an 'epoch' and after every epoch we replenish the number of Resos of a VM to the original allocated value. Any Resos left over from the earlier epoch are discarded. Deduction of Resos depends on the objectives of a pricing system. These objectives and pricing schemes are discussed in the section below.

D. Goals of Pricing

In general any pricing scheme should be able to regulate the supply and demand of the commodity. Therefore, when the price for a commodity is increased the demand for that commodity should fall. We use the same notion of pricing in ResEx when charging VMs for resource usage. In our current work we consider 2 goals for our pricing strategies, outlined below.

1) Maximize Resource Utilization: The pricing scheme should be able to allow VMs to purchase resources whenever they can and have enough Resos' to do so. Here the scheme will have fixed prices for a unit resource and is identical for all VMs. This allows each VM to use its resources to the maximum, and at a rate depending on the application(s) running within the VM. Once the VM does run out of Resos, its resource usage should be kept either to a minimum or completely blocked until the next epoch when its Resos may be regenerated. In this scheme, resource prices are set at the start of each epoch uniformly for all VMs, based only on the aggregate availability of and demand for resources.

2) Lower Latency Variation: Another goal of a pricing scheme is to allow each VM to pay for resources differently. The pricing differentiation starts when a VM starts using more of one resource causing other VMs to suffer in their performance – i.e., causes interference. The interference causes an increase in latency of the applications, which in turn may miss their SLA for delivery/receipt of data. Thus, interfering VMs will need to be charged more in order to reduce their demand for the resource and reduce congestion.

These and other pricing strategies require support for, dynamic monitoring of resource availability and usage (CPU and

I/O resources in our case), as well as for determining the 'interference' or 'congestion' caused by one VM, so that it can be charged for its resources in a differentiated manner, based on some priority notion or other policies.

VI. ResourceExchange Implementation

In this section we describe in detail the implementation of ResourceExchange or ResEx approach to lower I/O congestion for VMs using InfiniBand devices. First, we define how the Resos are distributed between all the VMs and the initial charging policy. Next, we describe 2 pricing policies that we developed based on the set of goals outlined at end of the previous section.

A. Charging VM Resource Usage in Resos

Since we are dealing with a multi-resource setup, Resos simplifies the way to charge VMs on resource usage. We first break down the CPU and I/O resources into indivisible quantities which can then be charged 'by the Reso'. ResEx performs allocation of Resos at every epoch, which in our case is 1 second. Resos for the resources consumed are deducted at every interval of 1 millisecond.

1) CPU Charging*:* ResEx converts the CPU cycles based on the frequency to a set of Resos for each CPU depending on the epoch/interval time. It then associates a VM with a CPU. We currently allocate an entire PCPU to each VM, thus the VM can use the entire CPU and the set of Resos associated with it. It charges every percentage of CPU the VM consumes in the interval. This percentage corresponds only to the CPU cycle count for the interval. Initially, the rate of charge is 1 Reso per CPU percent. Therefore, the VM is initially allocated $PercentPerInterval * NumberOfIntervals = 100 * 1000 = 100,000$ Resos for CPU consumption. Here the CPU percent is the indivisible quantity used for charging.

2) I/O Charging*:* ResEx allocates Resos based on the capacity of the InfiniBand. The physical capacity of our IB Link is 8Gbps (due to the 8b/10b IB encoding) i.e. 1 Gigabyte per second. From Section V-A we use MTUs as the smallest chargeable quanta. We assume a default MTU size of 1024bytes(1KB) used by applications to send data. Therefore, the IB Link is capable of sending $LinkBW/MTUSize = 1*1024*1024*1024/1024 = 1048576$ MTUs in one second or in one epoch. Initially, we charge 1 Reso per MTU sent. Here the entire Resos are shared between the VMs since the link is shared between them. ResEx can adjust this sharing to be equal between the VMs or unequal based on priority or weight of the VM.

We have now broken down each physical resource into their chargeable units. Each of these units can be charged using a single currency called Reso. Next, we describe how ResEx implements different charging policies based on the goals described in Section V-D

B. FreeMarket & Maximize Resource Utilization

In this scheme, every VM is charged based on the amount of I/O and CPU resources consumed. The rate of charging remains same for all VMs. For every interval and monitored VM, ResEx detects the number of MTUs sent by the VM (using IBMon) and the CPU cycles consumed (using the XenStat library). This is converted to a number of Resos to be deducted from the VM's allocation. Now, this value is deducted from the VM's Reso allocation. Thus, using this scheme every VM can use their maximum set of resources allocated in the epoch and this scheme achieves the maximum resource utilization pricing goal. We call this pricing scheme FreeMarket to denote that the VMs can freely purchase their resources.

However, when the VM's remaining Resos is below a certain limit (10% in our case), and more than 10% of the epoch is remaining, we start decreasing the CPU for that VM gradually. The CPU is decremented by 10% from its earlier allocated value. We do this to ensure that we do not have to abruptly stop the CPU for the VM when it runs out of Resos. This simply ensures a gradual decrease in performance experienced by the VM rather than a sudden stoppage. There are multiple ways in order to reduce the CPU when the VM runs out of Resos but those are beyond the scope of this paper.

Algorithm 1 shows the pseudo-code for the pricing scheme. The charging calls use the same rate of 1 Reso per unit resource for converting the resources used to Resos. The CPU cap returned depends on the number of Resos left of the VM allocation and is changed as described above.

Algorithm 1 FreeMarket Algorithm

```
1:  InitializeResos(FreeMarket)
2:  while $true$ do
3:      for all MonitoredVMs do
4:          $IBMTUs \leftarrow$ GetMTUs(ThisVMId)    ▷ Use IBMon
5:          $CPUPct \leftarrow$ GetCPUPercent(ThisVMId)
6:          $IBResos \leftarrow$ ConvertToIBResos(IBMTUs)
7:          $CPUResos \leftarrow$ ConvertToCPUResos(CPUPct)
8:          $CPUCap \leftarrow$ GetCPUCap(ThisVMId, IBResos, CPUResos)
9:          DecrementResos(ThisVMid, IBResos, CPUResos)
10:         SetVMCap(ThisVMId, CPUCap)
11:     end for
12: end while
```

C. I/O Shares & Lower Latency Variation

For meeting the second goal of pricing resources we now incorporate any latency feedback from the application within the VM in charging Resos. We base this charge on the 'Congestion Pricing' model where the VM(s) causing the congestion are charged more. Algorithm 2 describes the pseudo-code of the pricing scheme, where we now adjust the charging rate for the interfering VM when we detect that a VM is experiencing increased latencies. The line [6] in Algorithm 2 computes the

I/O interference percentage for the current VM. It looks at the latencies reported by the VM and computes the average and standard deviation of the values. The percentage increase in either of these is then returned to the scheme. If the percentage increase is greater than a certain value (i.e., SLA guarantee) then the interfering VM is found and its charging rate is increased based on the following formula:

$$Increase\ In\ Rate(r') = IOShare * IntfPercent$$

where IOShare is define as:

$$IOShare = \frac{MTUsSentByInterferingVM}{TotalMTUsSentByVMs}$$

Next, we compute the CPU cap to be computed for the current VM, decrement the Resos used and set the VM cap. When the interfering VM is being monitored then it will be charged with the new rate as computed in the reported VM in the last iteration of the loop. The CPU cap now for this interfering VM will be set as

$$New\ CPU\ Cap = \frac{100 * PreviousRate}{PreviousRate + r'}$$

Algorithm 2 I/O Shares Algorithm

```
1: INITIALIZERESOS(IOShares)
2: while $true$ do
3:     for all MonitoredVMs do
4:         $IBMTUs \leftarrow$ GETMTUS(ThisVMId)    ▷ Use IBMon
5:         $CPUPct \leftarrow$ GETCPUPERCENT(ThisVMId)
6:         $IOIntfPct \leftarrow$ GETIOINTF(ThisVMId)
7:         $IntfVMId \leftarrow$ GETIOINTFVMID(ThisVMId)
8:         $IOShare \leftarrow$ GETIOSHARE(ThisVMId, IntfVMId)
9:         $ChargeRate \leftarrow$ CHANGEIBRATE(IntfVMId, IOIntfPct)
10:        $IBResos \leftarrow$ CONVERTTOIBRESOS(IntfVMId, IBMTUs, ChargeRate)
11:        $CPUResos \leftarrow$ CONVERTTOCPURESOS(IntfVMId, CPUPct, ChargeRate)
12:        $CPUCap \leftarrow$ GETCPUCAP(ThisVMId, IBResos, CPUResos)
13:        DECREMENTRESOS(ThisVMid, IBResos, CPUResos)
14:        SETVMCAP(ThisVMId, CPUCap)
15:    end for
16: end while
```

VII. Experimental Evaluation

The testbed consists of two Dell PowerEdge 1950 servers. One of the servers has dual-socket quad-core Xeon 1.86 Ghz processors, while the other one has dual-socket dual-core Xeon 2.66 Ghz processors. Both servers have 4GB of RAM. Mellanox MT25208 HCA (memfull cards) cards are used in all machines, connected via a Xsigo VP780 10Gbps I/O Director Switch. We are using a Xen 3.3 unstable distribution on all servers. Also, all servers are running the same para-virtualized Linux kernel running in dom0. We are using para-virtualized InfiniBand driver modules which work under the Linux 2.6.18.8 dom0 and domU kernels. The guest OS' are configured with 512 MB of RAM and have the OFED-1.2 distribution installed. This distribution has been modified to run inside the guest VM. We are also running the OFED-1.2 distribution in the dom0s. Each guest domain is assigned a VCPU each in order to minimize the effects of shared CPUs.

We use certain terminologies in the experiments below which we explain first. We refer to an application running within a VM by its configured buffer size. For example, 64KB VM means that the BenchEx application uses 64KB as its buffer size. Interfering VM is a VM that has a buffer size greater than 64KB. The base case refers to a configuration where only one instance of the benchmark is run without an interfering load. In most of the experiments we use 2 VMs each on 2 separate physical machines unless otherwise mentioned. One of the VMs is the reporting VM and the other is the interfering VM. The latencies are always mentioned for the reporting VM.

To demonstrate the utility and effectiveness of ResEx we use different combinations and configurations of the BenchEx Latency benchmark. The results are classified by the goals and types of the pricing system developed.

A. *Maximization of Resource Utilization*

Here we compare the application performance of the FreeMarket algorithm with the base case and the case with interfering load. Figure 5 shows the change in the application latency when the FreeMarket algorithm is used. We can see that the latency of the 64KB VM (reporting VM) is lower when FreeMarket allocation is performed than the interfering case. This is due to the fact that the CPU cap is lowered for the 2MB VM periodically whenever its Reso count decreases below a minimum. This achieves some more control over the requests issued by the 2MB VM and thus reduces latencies for the 64KB VM.

Figure 6 shows the deduction in Reso when the FreeMarket algorithm is used. The algorithm keeps deducting Resos until a minimum level (10%) is reached after which it starts reducing the CPU Cap. The effect of this is seen by the 2MB VM. It also 'maximizes resource usage' by always allowing a VM access to the resource if it has enough Resos.

B. *Lower Latency Variation*

In Figure 7 we compare the application performance when the IOShares algorithm is used. We see that the algorithm is able to achieve near base case latencies for the application by taking into consideration the interference percentage of the 64KB VM and thus 'charging' the 2MB VM more for resources used. The CPU Cap is changed dynamically to a lower value for the 2MB VM by detecting the amount of congestion occurring. Here each VM reports its latencies to ResEx and on an average uses $10\mu s$ for reporting. This

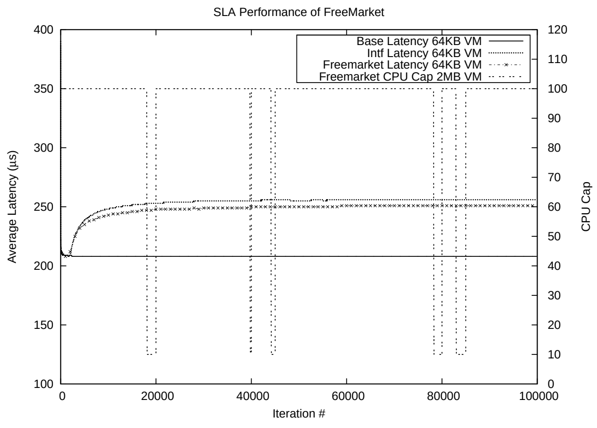

Fig. 5. Comparison of application latency when using FreeMarket algorithm.

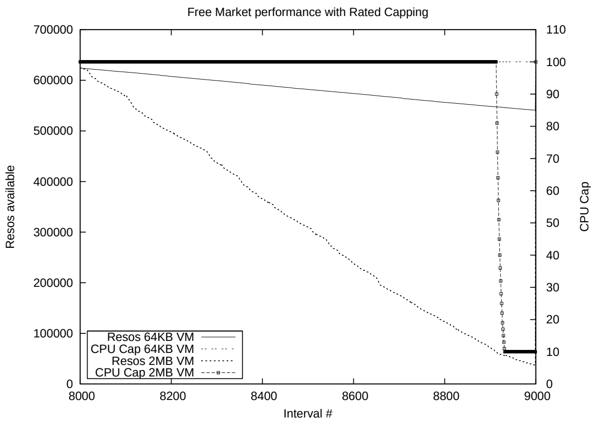

Fig. 6. Change in Reso during FreeMarket algorithm.

value for reporting is included with the latency to highlight the asynchronous nature of IB communication which may be useful in hiding latencies.

C. No Interference Handling

We also run two more cases where the interference between the 2 VMs is either negligible or the same as the reporting VM itself. Figure 8 shows the impact of those cases when either of the Reso management algorithms are used. In the figure, FM refers to the FreeMarket algorithm and IOS refers to the IOShares algorithm. The two cases chosen here are, one, when along with the Reporting 64KB VM there is another 64KB VM running BenchEx and two, the 2MB VM is issuing requests at 10 requests per epoch (a much slower rate than the interfering VM used in prior experiments).

Figure 8 compares the average latency experienced by the Reporting 64KB VM in these cases with the Base 64KB latency. We see that the values are almost equal to the Base values. This highlights two aspects of ResEx. One, ResEx can not only detect interference for a VM but also back off when there isn't any interference (in case of 64KB-2MB-nointf). Two, ResEx adapts to the I/O performed by the VMs to not penalize VMs if they are doing the same amount of I/O (in case of 64KB-64KB).

D. Response to Buffer Size

Applications may be configured to use different buffer sizes and therefore ResEx should be able to adapt the different resource management strategies based on that value. Figure 9 shows the latency of the 64KB VM when run against interference VM configured with different buffer sizes. We see that IOShares outperforms FreeMarket by maintaining the average latency very close to the base value.

FreeMarket does not limit the latency since it does not have access to that information. By allowing VMs to 'spend' their Resos as they wish FreeMarket does not eliminate congestion but makes sure the existing resources are used completely. This shows the 'work-conserving' capability of the FreeMarket algorithm. IOShares on the other hand, maintains the running average latency of the VMs and charges VMs

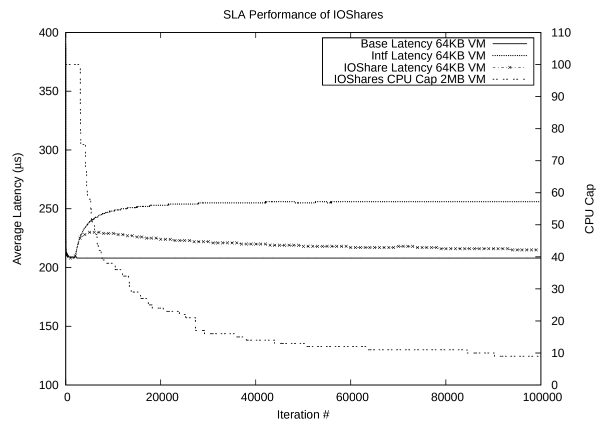

Fig. 7. Comparison of application latency when using IOShares algorithm.

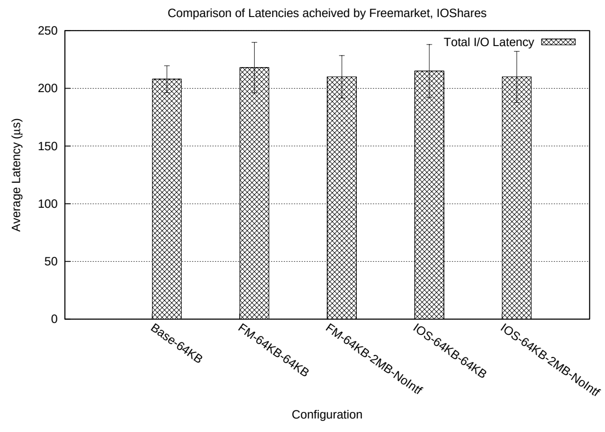

Fig. 8. Performance of FreeMarket and IOShares methods on non-interference cases.

causing congestion more. This leads to VMs resources being reduced slowly and as a result the congestion drops. Thus, IOShares is more conservative and aggressive than FreeMarket as a resource management scheme.

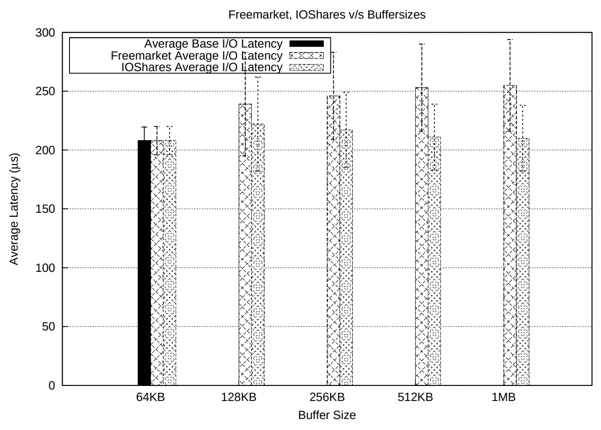

Fig. 9. FreeMarket and IOShares behavior with different interfering VM buffer size.

VIII. Related Work

In this section we briefly discuss existing research which is related to this paper.

Resource Management. There have been other recent efforts focused on resource management within virtualized environments for resources like power [15], storage [5], combined resource approaches [17] and InfiniBand resources [9] as well. Key difference with our work is that we are explicitly looking at the ability to support latency-sensitive applications in such environments.

Economics and Congestion Pricing. Finally, we are not the first ones to use congestion pricing for resource management. There has been earlier work in computer networks for congestion avoidance [10], energy management [14] that also used this model to reduce congestion. Also, it is used in QoS management [16] for providing better CPU scheduling for tasks. Currencies have also been used for providing energy management as shown in [23], [24]. In addition, market-based economic strategies are also used for resource allocation in large datacenters. This is very prevalent in cloud computing infrastructures since these models make it easier to allocate the physical infrastructure [2], [3].

IX. Conclusions

In this paper, we describe ResEx – a resource management approach which improves the ability of virtualized RDMA-based platforms to be shared by collocated latency-sensitive applications. ResEx uses the Resos resource unit abstraction and mechanisms based on supply-demand economic concepts and congestion pricing to dynamically manage VMs usage of I/O and CPU resources. By using a common "currency" ResEx allows us to establish conversion rates between the two types of resources, which is particularly necessary on RDMA-based platforms which support VMM-bypass, where the hypervisor's only mechanism for dynamically managing the VMs' I/O usage is through appropriate changes in their CPU allocations.

ResEx is implemented for InfiniBand platforms virtualized with the Xen hypervisor, and evaluated using an RDMA-based latency-sensitive benchmark BenchEx, modeled after a financial electronic exchange, and for two different resource pricing policies. The results demonstrate that virtualized RDMA-based platforms can be suitable for shared use by latency-sensitive applications. Furthermore, however, the mechanisms and abstractions introduced by ResEx are important for achieving better performance and management of SLA guarantees and for controlling unwanted latency variability.

References

[1] Bernt Arne degaard. C++ programs for Finance. http://finance.bi.no/ bernt/gcc_prog/.

[2] A. Byde, A. Byde, M. Sall, M. Sall, C. Bartolini, and C. Bartolini. Market-Based Resource Allocation for Utility Data Centers. Technical report, 2003.

[3] M. Feldman, K. Lai, and L. Zhang. A Price-Anticipating Resource Allocation Mechanism for Distributed Shared Clusters. In *Proceedings of the ACM Conference on Electronic Commerce*, June 2005.

[4] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of NDSS*, 2003.

[5] A. Gulati, I. Ahmad, and C. A. Waldspurger. Parda: proportional allocation of resources for distributed storage access. In *Proccedings of FAST*, pages 85–98, 2009.

[6] H. D. Henderson. Supply and Demand, 1922.

[7] I. Pratt and K. Fraser and S. Hand and C. Limpach and et al. Xen 3.0 and the Art of Virtualization. In *Ottawa Linux Symposium*, 2005.

[8] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2. www.infinibandta.org.

[9] M. Kesavan, A. Ranadive, A. Gavrilovska, and K. Schwan. Active CoordinaTion (ACT) - Towards Effectively Managing Virtualized Multicore Clouds. In *Cluster 2008*, 2008.

[10] P. Key, D. Mcauley, P. Barham, and K. Laevens. Congestion pricing for congestion avoidance. Technical report, 1999.

[11] M. Lee, A. S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik. Supporting Soft Real-Time Tasks in the Xen Hypervisor. In *Proceedings of VEE '10*, 2010.

[12] J. Liu, W. Huang, B. Abali, and D. K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. In *ATC*, 2006.

[13] Low-Latency 10-Gigabit Ethernet. http://myri.com/Myri-10G/documentation/Low_Latency_Ethernet.pdf.

[14] R. Nathuji, P. England, P. Sharma, and A. Singh. Feedback Driven QoS-Aware Power Budgeting for Virtualized Servers. In *Workshop on FeBID*, 2009.

[15] R. Nathuji and K. Schwan. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In *SOSP*, 2007.

[16] R. Neugebauer and D. McAuley. Congestion Prices as Feedback Signals: An Approach to QoS Management. In *ACM SIGOPS Workshop*, Sept. 2000.

[17] P. Padala, K. G. Shin, X. Zhu, M. Uysal, and et al. Adaptive control of virtualized resources in utility computing environments. In *SIGOPS*, 2007.

[18] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner. Achieving 10Gbps using safe and transparent network interface virtualization. In *VEE*, 2009.

[19] A. Ranadive, A. Gavrilovska, and K. Schwan. IBMon: Monitoring VMM-Bypass InfiniBand Devices using Memory Introspection. In *HPCVirtualization Workshop, Eurosys*, 2009.

[20] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan. Performance Implications of Virtualizing Multicore Cluster Machines. In *HPCVirtualization Workshop, Eurosys*, 2008.

[21] InterContinental Exchange. http://www.theice.com.

[22] Congestion Pricing. http://en.wikipedia.org/wiki/Congestion_pricing.

[23] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: managing energy as a first class operating system resource. *SIGOPS Oper. Syst. Rev.*, October 2002.

[24] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Currentcy: a unifying abstraction for expressing energy management policies. In *Proceedings of the USENIX Annual Technical Conference*, 2003.