A Novel Query Caching Scheme for Dynamic InfiniBand Subnets

Evangelos Tasoulas*, Ernst Gunnar Gran*, Bjørn Dag Johnsen†, and Tor Skeie*
*Simula Research Laboratory †Oracle Corporation
{vangelis, ernstgr, tskeie}@simula.no bjorn-dag.johnsen@oracle.com

***Abstract*—In large InfiniBand subnets the Subnet Manager (SM) is a potential bottleneck. When an InfiniBand subnet grows in size, the number of paths between hosts increases polynomially and the SM may not be able to serve the network in a timely manner when many concurrent path resolution requests are received. This scalability challenge is further amplified in a dynamic virtualized cloud environment. When a Virtual Machine (VM) with InfiniBand interconnect live migrates, the VM addresses change. These address changes result in additional load to the SM as communicating peers send Subnet Administration (SA) path record queries to the SM to resolve new path characteristics.**

In this paper we benchmark OpenSM to empirically demonstrate the SM scalability problems. Then we show that our novel SA Path Record Query caching scheme significantly reduces the load towards the SM. In particular, we show by using the Reliable Datagram Socket protocol that only a single initial SA path query is needed per communicating peer, independent of any subsequent (re)connection attempts.

I. Introduction

There is a lot of work going on, both in academia and the industry, to make cloud computing capable of offering High Performance Computing (HPC). With HPC-as-a-Service, traditional HPC users can save capital expenditure, while new user groups that cannot afford to own a private HPC cluster, can get on-demand access. However, the overhead imposed by virtualization combined with the extreme performance demands of HPC kept this idea immaterialized for a long time. During the last ten years, the situation has improved considerably as CPU overhead has been practically removed through hardware virtualization support [1], [2]; memory overhead significantly reduced by virtualizing the Memory Management Unit; storage overhead reduced by the use of fast SAN storages or distributed networked file systems; and network I/O overhead reduced by the use of device passthrough techniques like Single Root I/O Virtualization (SR-IOV) [3]. It is now possible for clouds to accommodate virtual HPC (vHPC) clusters using high performance interconnect solutions and deliver the necessary performance [4], [5], [6].

InfiniBand (IB) [7], the underlying network technology we focus on in this paper, is an HPC interconnection network technology offering high bandwidth and low latency. IB accelerates 224 HPC systems in the TOP500 supercomputers list as of November 2014 [8], 44.8% of the list. Each IB subnet requires a Subnet Manager (SM) which is responsible for the network initialization, topology discovery, path computation, and configuration of IB ports on switches and Host Channel Adapters (HCAs). In large subnets, the available paths between nodes grow polynomially and the SM becomes a potential bottleneck when many concurrent requests for path resolution are received. This scalability challenge is further amplified in a dynamic virtualized cloud environment, when a Virtual Machine (VM) with IB interconnect live migrates.

To ensure efficient virtualization, while maintaining high bandwidth and low latency, modern IB Host Channel Adapters (HCAs) support SR-IOV. However, it is challenging to support transparent live migration of VMs assigned to IB HCAs using SR-IOV [5], [6], [9]. Each IB connected node has three different addresses (LID, GUID, GID), that are further discussed in section II. When a live migration happens, regardless of the downtime attributed to the detaching of the passed through interface, that most of the existing works are focusing on, one or more of the IB addresses changes. Other nodes communicating with the VM-in-migration lose connectivity and try to find the new address to reconnect to by sending Subnet Administration (SA) path record[1] queries to the SM. The resulting futile communication in the underlying network towards the SM can be significant. In a large network, this message flooding towards the SM, caused by VMs migrating, will increase the overall network latency as the load on the SM increases.

In this paper, we first demonstrate the scalability challenges of the SM as the amount of SA queries increases. Next, our focus is to reduce the amount of SA queries in an IB subnet. We use the particular case of VM live migration, because many SA queries are generated by the peer nodes when a VM migrates and the IB addresses change. We implement a prototype and show that if we keep the same IB addresses after the migration, connections can be reestablished by sending less SA queries. Furthermore, we introduce a local SA path caching mechanism at the end nodes and show that subsequent SA queries after the initial connection is established, can be eliminated. The caching scheme is generic and when enabled, can alleviate the load of the SM with or without live migrations taking place. Our testbed is based on the OpenStack cloud platform and the caching scheme is implemented on the Reliable Datagram Socket (RDS) protocol [10] (version 3.1).

The rest of the paper is organized as follows: Section II gives background information on I/O Virtualization (IOV) and IB addressing schemes. In Section III we benchmark OpenSM to show the potential scalability issues of an SM that motivates our work further, while Section IV describes the design and

[1] A *path record* contains path characteristics like service level, MTU, etc. of a path between two communicating peers.

implementation of the given solution. The experiments and results are located in Section V, followed by the related work in Section VI. We conclude and add some future work in section VII.

II. Background

In this section we describe different Input/Output (I/O) Virtualization techniques with a particular focus on SR-IOV. The IB addressing schemes are presented, and the challenges leading to the increased amount of SA queries as a result of live migrations are discussed.

A. Network I/O Virtualization

I/O Virtualization (IOV) is needed to share I/O resources and provide protected access to them from the VMs. IOV decouples the logical device, which is exposed to a VM, from its physical implementation [11], [12]. Currently, there are two widespread approaches to IOV, both having their advantages and disadvantages:

a) **Software emulation** is a decoupled front-end/back-end software architecture. The front-end is a device driver placed in the VM, communicating with the back-end implemented by the hypervisor to provide I/O access. Two well known implementations of software emulation IOV are 1) *Emulation of real devices* and 2) *Paravirtualization*. When emulating real devices, a VM can run unmodified drivers of the emulated device, and the guest OS behaves as if it was running on real hardware. However, emulating real devices suffer from poor performance and adds overhead to the hypervisor[2], because CPU cycles have to be dedicated to emulate non-existing hardware. Paravirtualization exposes a lightweight optimized virtual hardware to the guest OS, that improves performance and reduces the overhead compared to the emulation of real devices method. The disadvantage is that special drivers for the virtual hardware need to be installed in the guest OS. With the software emulated techniques, the physical device sharing ratio is high and live migration is possible with just a few milliseconds of network downtime.

b) **Direct device assignment** involves a coupling of I/O devices to VMs, with no device sharing between VMs. Direct assignment or device passthrough provides near to native performance with minimum overhead. The physical device is directly attached to the VM, bypassing the hypervisor, and the guest OS can use unmodified drivers. The downside is limited scalability, as there is no sharing; one physical network card is coupled with one VM. *Single Root IOV (SR-IOV)* allows a physical device to appear through hardware virtualization as multiple independent lightweight instances of the same device. These instances can be assigned to VMs as passthrough devices, and accessed as Virtual Functions (VFs) [3]. SR-IOV eases the scalability issue of pure direct assignment. Unfortunately, there is currently no easy way to live-migrate VMs without a network downtime in the order of seconds when using direct device assignment [13].

[2]The software component responsible for managing VMs, is called Virtual Machine Monitor (VMM) or hypervisor [11].

B. The InfiniBand Addressing Schemes

InfiniBand uses three different types of addresses [7], [14], [9]. First is the 16 bits Local Identifier (LID). At least one unique LID is assigned to each HCA port and each switch by the SM. The LIDs are used to route traffic within a subnet. Since the LID is 16 bits long, 65536 unique address combinations can be made, of which only 49151 (0x0001-0xBFFF) can be used as unicast addresses. Consequently, the number of available unicast addresses defines the maximum size of an IB subnet.

Second is the 64 bits Global Unique Identifier (GUID) assigned by the manufacturer to each device (e.g. HCAs and switches) and each HCA port. The SM may assign additional subnet unique GUIDs to an HCA port, which is particularly useful when SR-IOV VFs are enabled.

Third is the 128 bits Global Identifier (GID). The GID is a valid IPv6 unicast address, and at least one is assigned to each HCA port and each switch. The GID is formed by combining a globally unique 64 bits prefix assigned by the fabric administrator, and the GUID address of each HCA port.

C. SR-IOV, InfiniBand and Live Migrations

Providing HPC as an elastic and efficient cloud service is challenging due to the added overhead of virtualization. When it comes to the interconnection network, none of the software emulation approaches from section II-A are suitable. HPC interconnection networks rely heavily on hardware offloading and bypassing of the protocol stack and the OS kernel to efficiently reduce latency and increase performance [15]. Thus, the only viable option to provide high performance networking in VMs, is to use a direct device assignment technique. To still be scalable, we, as others working with IB and virtualization [4], [6], [9], chose to use SR-IOV for our experiments.

Unfortunately, direct device assignment techniques pose a barrier for cloud providers if they want to use transparent live migrations for data center optimization. The essence of live migration is that the memory contents of a virtual machine are copied over to a remote hypervisor. Then the virtual machine is paused at the source hypervisor, and its operation resumed at the destination. When using software emulation methods, the network interfaces are virtual so their internal states are stored into the memory and gets copied as well. Thus, the downtime is in the order of a few milliseconds [16]. In the case of direct device assignment like SR-IOV VFs, the complete internal state of the network interface cannot be copied as it is tied to the hardware [5]. The SR-IOV VFs assigned to a VM will need to be detached, the live migration will run, and a new VF will be attached at the destination. This process will introduce downtime in the order of seconds.

In the case of InfiniBand and SR-IOV, the downtime associated with live migrations has been discussed by Guay et al [9], [17]. However, there is no related work discussing the imposed scalability challenges faced by the SM. When a

VM using an IB VF is live migrated, a clear impact on the underlying network fabric and the SM will be introduced, due to a change of all the three addresses of the VM [9]; The LID will change because the VM is moved to a different physical host with a different LID. The virtual GUID (vGUID) that is assigned by the SM to the source VF will change as well, as a different VF will be attached at the destination. Subsequently, since the vGUID is used to form the GID, the GID will change too. As a result, the migrated VM will suddenly be associated with a new set of addresses, and all communicating peers of the VM will start sending concurrent SA path record query bursts to the SM, trying to reestablish lost connections. These queries are causing extra overhead to the SM, and supplementary downtime as a side effect. If the migrated nodes communicate with many other nodes in the network, the SM can become a bottleneck and hamper overall network performance.

III. Motivation

In large IB subnets the SM is in general a potential bottleneck due to its centralized design [18]. A main concern is the amount of SA queries the SM needs to handle, and particularly the number of path record requests. The amount of SA queries increases polynomially as the number of nodes in the network increases. As the scalability issues of the SM are not very well documented in the literature, we benchmark OpenSM to empirically demonstrate how the SM is challenged even by a relatively low amount of SA queries.

A. Testbed

Our testbed consists of 3 SUN Fire X2270 servers with 4 cores and 6 GB RAM each; 4 HP ProLiant DL360p Gen8 servers with 8 cores (two CPU sockets) and 32 GB RAM each; 2 HP ProLiant DL360p Gen8 servers with 4 cores and 32 GB RAM each; and 2 InfiniBand SUN DCS 36 QDR switches. The OpenStack Grizzly cloud environment is deployed on Ubuntu 12.04, and a CentOS 6.4 image is used for the virtual machines. The three SUN Fire servers are used as the OpenStack Controller, Network and Storage nodes. The HP machines that serve as OpenStack compute nodes, are referred to as *flooder nodes* while benchmarking. The OpenStack management network is based on Ethernet, while IB is used for the VMs. All compute nodes are equipped with the Mellanox ConnectX®-3 VPI adapters [19] and SR-IOV enabled Mellanox OFED V2.0 drivers. The same version of Mellanox OFED is also installed on the CentOS virtual machines. A reference summary of the hardware can be found in Table I.

Qty	Hardware	Usage
3x	SUN Fire X2270 6GB RAM 4 CPU Cores	Openstack Controller, Network, Storage.
2x	HP ProLiant DL360p 32GB RAM 4 CPU Cores Mellanox ConnectX®-3 VPI HCA	OpenStack Compute. One of the nodes is running the SM and LIDtracker in section IV.
4x	HP ProLiant DL360p 32GB RAM 2x4 (8) CPU Cores Mellanox ConnectX®-3 VPI HCA	OpenStack Compute (Used independently for the SM benchmarking in section III).
2x	SUN DCS 36 QDR InfiniBand switches	InfiniBand interconnection. Running the SM in section III.

TABLE I
TESTBED HARDWARE

Algorithm 1 Experiment Template

```
1:  procedure EXPERIMENTTEMPLATE(m, n, k)
2:      Spawn SAF Instance On SIF
3:      Collect Data for 20 seconds
4:      for r = 2; r <= m; r++ do //for each new round
5:          for x = 1; x <= n; x++ do //...and each MIF
6:              for i = 1; i <= k; i++ do //spawn flooders
7:                  Spawn SAF Instance i On MIF_x
8:              end for
9:          end for
10:         Collect Data for 20 seconds
11:     end for
12: end procedure
```

B. Benchmarking Procedure

Considering the size of our testbed, we run OpenSM in one of the IB switches, using *opensm-3.2.6_20130819-0.1_oracle_patch_11.9* with SR-IOV support.

An application, *SA Flooder* (SAF), was written to flood the SM with SA path record queries. The application sends SA queries synchronously, meaning that for each query sent, SAF waits for the reply, or a timeout, before sending the next one. For each query, the response time is logged. As the SM becomes saturated, the response time is expected to increase, and the amount of SA queries served reach an upper limit. Because SAF is synchronous, we spawn multiple instances in the flooder nodes to increase the concurrency.

Because multiple threads introduce local interference as they compete for local resources, the response time logged by the local instances might deviate from the actual response time of the SM. To avoid unreliable measurements due to local interference, in all of the experiments, one of the flooder hosts is running only a single SAF instance at all times. This flooder host, the *Single Instance Flooder* (SIF), is used to measure response time and generate plots, while the remaining flooder hosts, the *Multiple Instance Flooders* (MIFs), are used to push the SM to its limits as an increasing number of SAF instances are spawned on the MIFs.

We ran experiments using the template in Algorithm 1, and varied the following parameters:

1) The number of MIFs (n): The number of MIF hosts participating in a given experiment influences the amount of SA queries we can potentially push towards the SM. $n \in \{0, 1, ..., 5\}$, as one out of six flooder nodes is always acting as the SIF.

2) The instance spawn number (k): Each MIF in the experiment will spawn k new SAF instances each *round* (see below) to increase the load on the SM.

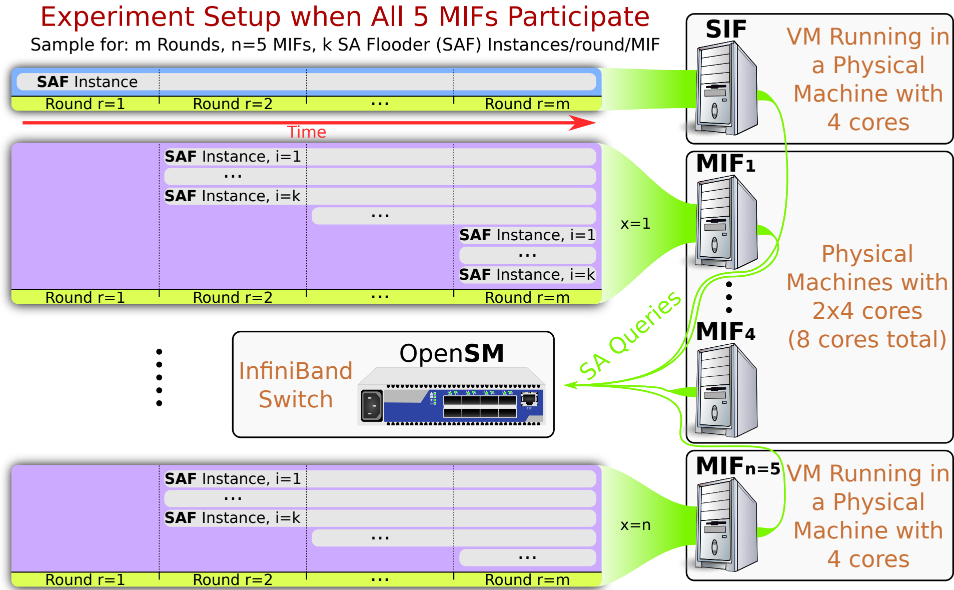


Fig. 1. Experiment Setup

SAF running on	Single thread Max SA Queries/sec	Multiple threads Max SA Queries/sec
physical node, 4 cores	~250	~1000
physical node, 8 cores	~250	~2100
virtual node, 4 cores	~440	~1150
virtual node, 8 cores	~440	~1100

TABLE II
SAF PERFORMANCE ON SINGLE NODES

SAF Node name	Real host mapping
SIF	Running on VM on top of a Physical machine with 4 cores
MIF1-MIF4	Running on Physical machine with 8 cores
MIF5	Running on VM on top of a Physical machine with 4 cores

TABLE III
FLOODER HOST NAME MAPPINGS TO PHYSICAL HOSTS

3) The number of rounds (m): Each *round* starts with an initial phase where each of the n MIFs spawn k new SAF instances. Subsequently, measurements are collected for 20 seconds to ensure proper statistical significance, before a new round starts. Note that the SIF is always left untouched, and in particular, that during the first round, only the SIF is active. The m rounds define the length of an experiment.

C. Node Roles in the Benchmark

As shown in section III-A, two flooder nodes are equipped with a quad core CPU, while four nodes are equipped with two quad core CPUs. The CPU model, as the rest of the hardware, is identical. After preliminary experiments we observed that a physical machine equipped with eight CPU cores has slightly more than twice the capability of a four core machine to push SA queries. Interestingly, when we pushed SA queries from a four core VM with an IB VF attached, the VM outperformed its physical counterpart, the host machine. However, when comparing an eight core VM with its physical counterpart, the dual-socket eight cores host, the VM performance drops even below that of a four core VM. This performance reduction may be attributed to the virtual-to-physical CPU affinities [20].

Table II shows the results of the preliminary experiments. The numbers in the second column show the max performance when only a single undisturbed SAF instance is running on a host. The numbers in the third column show the max performance a single host can achieve when multiple SAF instances run on the host. We refer to as *max performance*, the max SA queries generation capability of a flooder node. According to Table II and the SIF/MIF concept we introduced in section III-B, to maximize the number of total SA queries per second, the flooder hosts were assigned the roles given in Table III. The four physical machines with eight physical cores ran the SAF natively. The two physical machines with four physical cores ran the SAF in a VM, since the VMs with the SR-IOV VFs deliver higher flooding rate in the four core machines. The SIF host is running in one of the two VMs while the remaining hosts are used as MIF hosts (MIF1-MIF5). Fig. 1 shows the experiment setup and the type of hardware assigned on each flooder node (SIF/MIFs) when all flooder nodes participate in the experiment.

D. Benchmarking Results

By combining Table II and Table III we can induce the max SA queries per second generation capability for each of the flooder hosts, as given in Table IV. Moreover, the expected theoretical maximum when all of the SIF and MIF hosts send SA queries simultaneously, should be the summation of all

SAF Node name	Max SA Queries/sec
SIF	~440
MIF1-MIF4	~2100
MIF5	~1150

TABLE IV
SIF/MIF HOSTS MAX SA QUERIES/SEC CAPABILITY

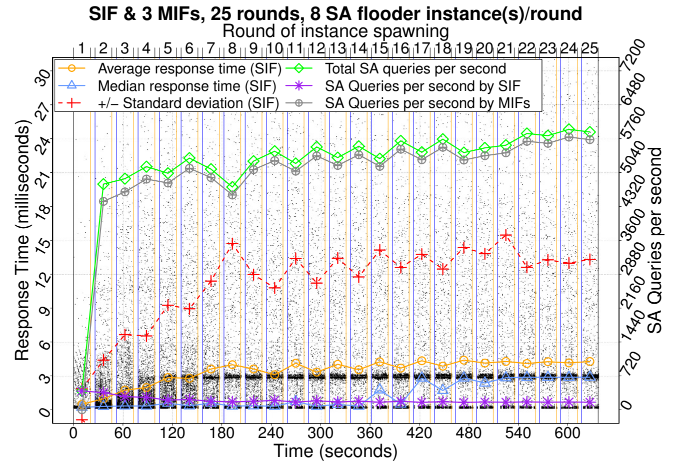
flooder hosts' SA queries per second generation capabilities, as given by the equation 1.

$$SIF_{max} + \sum_{x=1}^{5} MIFx_{max} \approx 440 + 1150 + 2100 \cdot 4 \approx 9990 \quad (1)$$

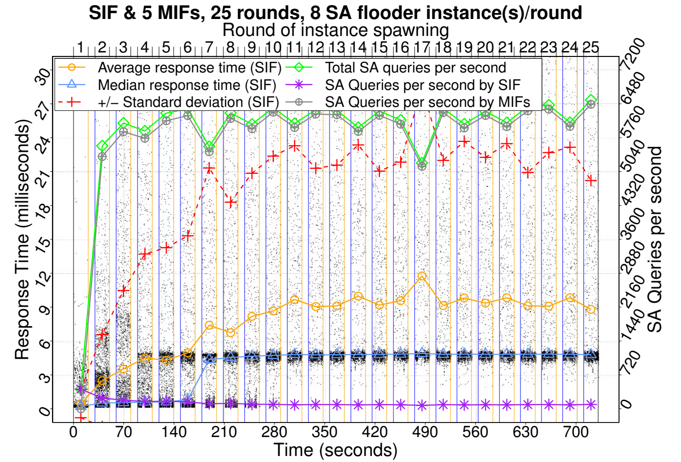
A scalable SM should be able to handle all SA queries, and the response time should be kept low, even as the number of total SA queries/sec in the subnet increases. To observe the behavior of the SM, we ran experiments while we slowly increased the total amount of SA queries by increasing the sending concurrency on each flooder host, and the number of flooder hosts participating in the experiment. Then we observed when the SM was saturated. In the first experiment only the SIF host and one MIF host ($n = 1$) was participating, while on each subsequent experiment, one more MIF host was added to increase the total amount of SA queries towards the SM. In the last experiment with one SIF and five MIF ($n = 5$) hosts the SM should be able to serve around 10000 SA queries per second according to equation 1.

We present the results after running each experiment for $m = 25$ rounds and $k = 8$ instances per round for $n \in \{1, 2, ..., 5\}$ MIFs. Recall that all of the n participating MIFs spawn k new SAF instances at the beginning of each round. Note that in all experiments, although $n \in \{1, 2, ..., 5\}$, $n = 0$ always and only for round one (refer to Algorithm 1 and Fig. 1). In the final round (25th) of the last experiment with $n = 5$ MIFs, 961 total instances are sending SA queries with these settings (high concurrency). For the experiment that the SIF and all MIFs ($n = 5$) are participating, we also ran a test when only one SAF instance per round ($k = 1$) is spawned on each of the MIF hosts to show the difference when the concurrency changes. In this case, the final round (25th) has 121 total instances running (reduced concurrency).

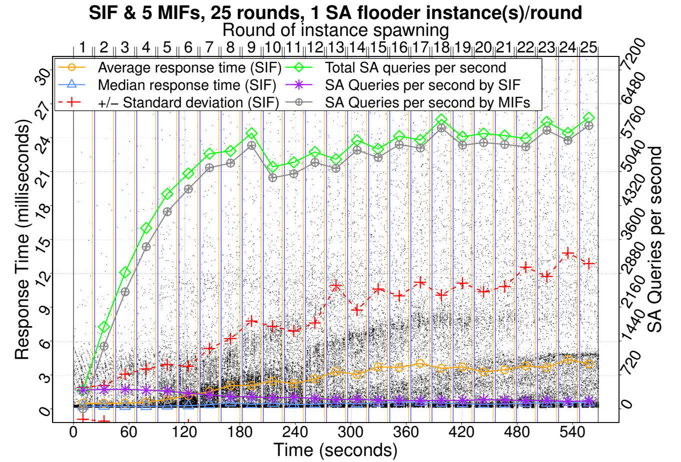
In Fig. 2 the results from three of the experiments showing the SM performance trends on different rounds are illustrated, and on Table V the numbers acquired and the expected theoretical values are presented for all six of the experiments. The average response time, the median response time and the standard deviation recorded from the SIF host are plotted on the left y axis of Fig. 2. The queries per second that the SIF and MIF hosts send, as well as the sum of these values (that gives the total serving capability of the SM), are plotted on the right y axis. The noise-like dots on the plots are individual SA query requests, which are the source of the calculation for the rest of the statistics plotted. In all of the plots, at the first round only the SIF host is running, and the first round of instance spawning at the MIF hosts starts on round two. The solid vertical lines separate the different rounds of the SAF instance spawning. The short gaps between rounds of SAF instance spawning, where no dots are plotted at all, is the time that it takes to spawn all the SAF instances in all of the participating hosts. This is the reason why the gap is larger when more MIF hosts n with increased concurrency k



(a) SIF & MIF1-MIF3: $n = 3, k = 8, m = 25$



(b) SIF & MIF1-MIF5: $n = 5, k = 8, m = 25$



(c) SIF & MIF1-MIF5: $n = 5, k = 1, m = 25$

Fig. 2. OpenSM benchmarking results

	1st round SIF results				25th (final) round SIF results				25th round total SA Queries/s	
Participating flooder nodes	**Q/s**	**Mean resp.time**	**Median resp.time**	**Std.dev.**	**Q/s**	**Mean resp.time**	**Median resp.time**	**Std.dev.**	**Measured**	**Expected theoretical max**
SIF & 1 MIFs	396	0.45ms	0.22ms	1.20ms	435	0.50ms	0.22ms	2.20ms	2605	~2540
SIF & 2 MIFs	405	0.47ms	0.22ms	1.70ms	174	1.65ms	0.24ms	5.76ms	4555	~4640
SIF & 3 MIFs (Fig. 2a)	398	0.48ms	0.22ms	1.36ms	161	4.27ms	2.89ms	9.03ms	5900	~6740
SIF & 4 MIFs	406	0.48ms	0.22ms	1.47ms	98	8.04ms	4.10ms	10.60ms	6285	~8840
SIF & 5 MIFs (Fig. 2b)	417	0.45ms	0.22ms	1.24ms	92	8.80ms	4.79ms	11.40ms	6561	~9990
SIF & 5 MIFs (Fig. 2c)	397	0.49ms	0.22ms	1.41ms	167	3.99ms	0.53ms	8.87ms	6017	~9990

TABLE V
OVERALL SYSTEM PERFORMANCE

participate on each experiment. It takes more time to deploy the SAF instances in all of the hosts on each round. The gaps are not included in the calculations, since they are interim, not fully populated round states.

The test where $n = 1, k = 8$ and only the SIF and MIF1 hosts participate in the experiment is not plotted, but the SM can easily handle 2605 SA queries per second as seen in Table V. The performance of the SIF host is stable from the beginning until the end of the experiment and the observed value follows the theoretical max value as expected. Similarly, when $n = 2, k = 8$ the MIF2 host is added in the experiment as well, and the SM can handle 4555 SA queries per second. The response times are still not changing much until the end of the experiment, but the rate that the SIF host is served on final round is already much lower, 174 Q/s, showing some first signs of saturation. The observed value still follows the theoretical max value. In Fig. 2a, $n = 3, k = 8$ and MIF3 host is added in the list of flooder hosts that participate in the experiment. This is the first time that the measured value for the total SA queries served per second, 5900, does not follow the theoretical value, 6740. Also the mean, median and standard deviation of the response times are substantially increasing. Still, as seen in Fig. 2a, on each subsequent round that the concurrency is increased as more SAFs are spawned, the total number of SA queries served is increased as well. When MIF4 and MIF5 ($n = 5, k = 8$) are added in the experiment as seen in Fig. 2b (row five in Table V), the SM cannot keep up on serving more SA queries and the median, average, and standard deviation values are increasing while the SA queries pushed by the SIF host are decreasing to 92 Q/s. The peak serving performance of the SM is also fluctuating, showing signs of instability as shown by the standard deviation. In Fig. 2c we demonstrate what happens when there are less SAF instances in the system since only one new instance is spawned on all 5 MIF hosts per round ($n = 5, k = 1$). Less SAF instances means reduced concurrency in the SA query requests. The SM is still saturated, but the overall system is more stable and responds faster in Fig. 2c even when compared with Fig. 2a. In particular, the median response value, 0.53ms, is significantly lower.

We conclude that the maximum capacity of OpenSM when running on our IB switch is around 6000 SA queries per second, and that its behavior can get highly unstable even before reaching this saturation point if the concurrency of the received requests is high. Of course, the CPU and memory of the switch are not so powerful (1.10GHz Z510 Intel Atom CPU and 512 MB RAM), and consequently OpenSM's performance is affected. However, even in a large subnet with a powerful SM node, when tenths of thousands of nodes are present, the amount of SA queries will increase polynomially when nodes try to communicate with each other. Moreover, in a large subnet the SA path query responses take more time, because the paths between hosts are longer with more intermediate hops and links. The SM has to traverse the path between the two nodes to get information such as the minimum supported MTU and link speed for both the forward and reverse paths.

It is reasonable to anticipate that if we had more MIF nodes or a large and busy subnet, the performance of the SIF would drop even more in our experiments. If the SIF was a real application with certain time-bounded Quality of Service (QoS) requirements, the SM would pose a potential bottleneck for the application. In the context of virtualization if a server with thousands of connected peers is migrated, a burst of thousands of SA Queries will be generated towards the SM on top of the already existing load as explained in section II-C, and analyzed with experiments in the rest of the paper.

IV. DESIGN AND IMPLEMENTATION

In this section we discuss in detail our SA path record caching mechanism. Furthermore, we describe the implemented prototype that allows us to migrate VMs with their associated IB addresses, in order to make use of our novel caching scheme before and after the migration.

A. Prototype Design

The testbed used for our experiments is described in section III-A. In our prototype, we use OpenStack to perform VM live migrations of VMs with IB SR-IOV VFs attached. We use Remote Direct Memory Access (RDMA) over the RDS protocol to reestablish the communication after the migration. For the prototype we changed OpenStack, OpenSM and the RDS Linux kernel module. In addition, we created a program that we called *LIDtracker*. *LIDtracker* keeps track of the IB addresses associated with each VM and orchestrates the migration process. The current prototype work flow is as follows:

i) LIDtracker enables OpenSM's option *honor_guid2lid_file*.
ii) The file *guid2lid*, generated by OpenSM, is then parsed by LIDtracker and sorted by GUID in ascending order.

LIDs are assigned to the GUIDs starting from one. We call these LIDs *base LIDs* for the physical hosts.

iii) All of the IB enabled OpenStack compute nodes are scanned for running VMs. Each VM is assigned a LID in decreasing order, starting from 49151 (the topmost unicast LID). We call these VM LIDs *floating LIDs*.

iv) The *floating LIDs* replace the *base LIDs* in the OpenStack compute nodes where VMs are running. Because Mellanox CX3 adapters use the SR-IOV Shared Port model [21], the hypervisor shares the LID with the VMs. Due to this limitation in the Shared Port model, we only support one VM running per hypervisor in the current prototype, and a VM can only be migrated to a hypervisor where no other VM is currently running.

v) When a migration for VM_x is ordered from the OpenStack API, the SR-IOV VF will be detached from the VM, otherwise the migration cannot start. When the device removal is completed and the migration is in progress, OpenStack will notify LIDtracker that VM_x is moving from $Hypervisor_y$ to $Hypervisor_z$. LIDtracker will then change the LID of $Hypervisor_y$ back to its *Base LID* and $Hypervisor_z$ will get the *floating LID* associated with VM_x. LIDtracker will also assign the vGUID associated with VM_x to the next available SR-IOV VF at $Hypervisor_z$. During the migration, the VM has no network connectivity.

vi) LIDtracker will restart OpenSM to apply the changes.

vii) When the migration is completed, OpenStack will add the next available SR-IOV VF to VM_x on $hypervisor_z$ and the VM will get back its network connectivity. The VM is exposed to the same IB addresses (LID, vGUID and GID) that it had before the move. From the perspective of the VM, it appears like the IB adapter was detached for the time needed to migrate and the same IB adapter was reattached since the addresses did not change.

B. SA Path Record Caching

The SA Path Record caching mechanism has been implemented in the RDS Linux kernel module and OpenSM.

1) OpenSM Modifications: OpenSM has been modified in order to signal the clients if the SA path caching should be enabled in the IB subnet or not. If caching is enabled the SA path record requests will be greatly reduced in the IB subnet. However, we have to make sure that when a live migration happens the migrated VMs will get their IB addresses migrated as well, so LIDtracker should be running in the network. If LIDtracker is not running and caching is enabled, after the live migration of a VM has finished, the peers will not be able to reconnect because the migrated VM will get a different set of addresses but the peers will still be trying to reconnect to the old cached addresses. A boolean option *subnet_supports_sa_path_caching* was introduced in OpenSM to configure a subnet that supports address caching. When we run OpenSM with the option *subnet_supports_sa_path_caching*, the reserved field (with bit offset 353 [7]) in the SA Path Record Response is used to raise the caching flag.

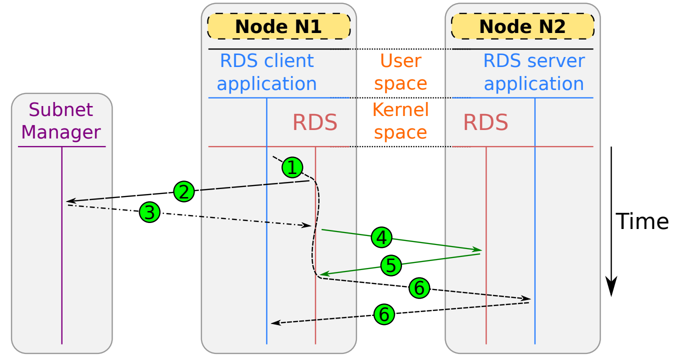


Fig. 3. RDS initializing connection

2) RDS Protocol Operation: Since our implementation is using RDS to demonstrate our findings, it is necessary to explain how the original RDS protocol establishes a connection between two hosts. First, IP over IB (IPoIB) needs to be set up in all of the communicating peers. At the beginning, RDS will use the IPoIB IP address of a specific IB port to determine the GID address of the port. After the GID address is resolved, RDS has enough information to perform the necessary path record lookups and establish the IB communication.

In Fig. 3, the client side of an upper layer application runs in *Node N1* and the server side of the application runs in *Node N2*. The client side of the application creates an RDS socket and tries to communicate with the server (Fig 3.①). RDS will send an SA Path Record request to the SM from *Node N1* (Fig 3.②), get a response (Fig 3.③), and try to initiate a connection with *Node N2* by sending a connection request (Fig 3.④). If the connection is successful, RDS will establish a communication channel (Fig 3.⑤) with an *RDMA_CM_EVENT_ESTABLISHED* event in both sides and the upper layer application can communicate (Fig 3.⑥). If anything goes wrong at the time of the initial connection, RDS on the client side (*Node N1*) will retry to establish a connection with a random backoff mechanism. The server is not yet aware of the intention of the client to communicate. If anything goes wrong after the connection has been established, both of the RDS sides (the client and the server from an application perspective) will actively engage a reconnection with the peer. The random backoff mechanism in the connection process is useful to avoid race conditions when both sides are engaging a connection, as illustrated in Fig. 4.

In Fig. 4, there is an ongoing communication between *Node N1* and *Node N2* (Fig. 4.①) when the connection drops (Fig. 4.②). Both RDS ends realize that the connection is down and wait for some random time (Fig. 4.③) before they try to reconnect by sending an SA path record request to the SM (Fig. 4.④). After the SA path record response is received (Fig. 4.⑤), a connection request will be sent (Fig. 4.⑥).

In the illustrated case in Fig 4, the random backoff time chosen by the two nodes in step ③ was almost the same. Thus, even though *Node N2* got the SA path record response slightly faster than *Node N1* and tried to initiate the connection first

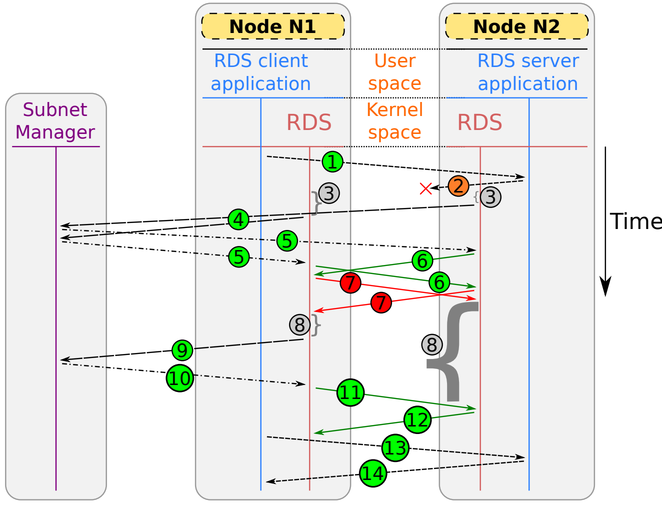


Fig. 4. RDS reestablishing connection

on step ⑥, the connection request did not reach N1 before N1 sent a connection request itself. In this case, both of the RDS ends have an outstanding connection request. Then, when they receive the connection request from their peer, they reject it, as seen in step ⑦. In step ⑧, the two nodes chose a random delay time once more before they retry to reconnect. This time the random backoff time chosen by *Node N2*, is significantly larger than the one chosen by *Node N1*. N1 gets the priority and repeats the connection establishment process; sends an SA path record request (Fig. 4.⑨), gets back a response (Fig. 4.⑩), sends a connection request (Fig. 4.⑪), and the connection request reaches *Node N2* before N2 tries to initiate a connection itself. N2 accepts the incoming connection as shown in Fig. 4.⑫. The communication then resumes for the upper layer application in steps ⑬ and ⑭.

Note that RDS will not create a communication pair for each upper layer application. Instead, it will create a communication pair between any two nodes in a network that communicates, and it will actively try to reconnect if the connection drops. All upper layer RDS applications at a node communicating with a given peer node, share the same communication pair.

3) RDS Protocol Modifications: Our novel SA Path Record Caching mechanism is implemented in the RDS protocol and the caching table is stored in the memory of each node. The algorithm used for the caching is presented in Algorithm 2. The first time a source host (SHost) tries to communicate with a destination host (DHost), the SHost will send an SA Path Record Request to the SM. If the response has the caching flag raised (explained in section IV-B1), the SHost will use a local caching table to store the path characteristics associated with the given GID address of the DHost (DGID). Moreover, the SHost now knows that caching is supported by the SM, so the next time the SHost will try to (re)connect with any DHost, it will lookup in the caching table first. If the path information for the given DHost is found, there will be no SA query sent to the SM, and SHost can instantly attempt to connect with the DHost.

Algorithm 2 SA Path Caching on RDS

1: **private bool** $SA_PathCachingEnabled$
2: **private list** $SA_PathRecordCacheTable$
3:
4: **procedure** RDSMODULEINITIALIZATION
5: // The Caching table is initialized
6: $SA_PathRecordCacheTable = empty$
7:
8: // We do not know yet if SA Path Caching is
9: // enabled by the SM, so we assume not.
10: $SA_PathCachingEnabled = False$
11: **end procedure**
12:
13: **procedure** (RE-)CONNECTIONESTABLISHMENT($DGID$)
14: **struct PathRecord** $DST_Path = NULL$
15:
16: // Use the cache only if the SA Path Caching is
17: // enabled by the SM
18: **if** $SA_PathCachingEnabled$ **then**
19: **if** $DGID$ in $SA_PathRecordCacheTable.$
20: $DGIDs$ **then**
21: $DST_Path = Cached_PathRecord$
22: **end if**
23: **end if**
24:
25: // If DST_Path is $NULL$ at this point, either the
26: // cache is disabled by the SM, or the path
27: // characteristics for the host with the given $DGID$
28: // have never been retrieved. In any case, we need to
29: // send a PathRecord Query to the SM.
30: **if** $DST_Path == NULL$ **then**
31: $SendAnewSA_PathRecordQueryToTheSM$
32: $WaitForTheReply$
33: $DST_Path = PathRecordResponse$
34:
35: // If caching is enabled by the SM the reply will
36: // have the reserved field in the PathRecord set
37: // to 1. If not, the reserved field is 0
38: **if** $DST_Path \rightarrow Reserved_Field$!= 0 **then**
39: $SA_PathCachingEnabled = True$
40:
41: // Insert the DST_Path in the caching table
42: $SA_PathRecordCacheTable$.append(
43: DST_Path)
44: **end if**
45: **end if**
46: $connect_to(DST_Path)$
47: **end procedure**

If compared with the original RDS protocol, the initial connection attempt described in Fig. 3 is not changing. That is, initially the path characteristics are not known, so one Path Record query is unavoidable. However, with reference to the interrupted connection in Fig. 4, no SA queries need to be sent to the SM. In the case described in Fig. 4, the steps ④, ⑤, ⑨ and ⑩ are eliminated, thus, the connection re-establishment is faster and the load towards the SM is lower.

The overhead of the caching scheme is small, as 568 bits[3] (71 bytes —the size of the *struct ib_sa_path_rec*—) are needed

[3]The size of a PathRecord query response is 512 bits (and 51 bits are reserved), but in the implementation some elements that are defined as e.g. 1 bit, are encoded in 8 or 32 bit long variables.

for each cache entry on each node. In the worst case scenario where a node in a fully populated IB subnet with 49151 nodes is communicating with all other nodes (49150 nodes), $49150 \cdot 71bytes = 3489650 \approx 3.33MiB$ of extra memory is needed on this node.

V. Experiment Results and Evaluation

In this section we present and evaluate the experimental results. We show the increased amount of SA queries as a result of the IB addresses change when live migrating. By using our prototype, we demonstrate the improvements achieved when the addresses are kept and the cache is used after the migration.

A. Experiment Process

After the benchmark we performed in section III, it becomes evident that in an IB subnet, increased interaction with the SM can substantially decrease the performance of the network. In a large subnet with thousands of nodes, even if only one additional SA query is sent from each node, the SM will end up being flooded with thousands of messages. When live migrations take place in a dynamic IB-based cloud, many excessive SA queries will be sent, as explained in section II-C. The amount of SA queries per second per node that is sent as a result of a live migration is an application and workload specific parameter, and in our case we use RDS to demonstrate how significant the issue can become in a large subnet.

In our experiments, first, we show the additional amount of SA queries generated when a VM is migrated and the IB addresses change at the destination. In this scenario, the peers that communicate with the migrated VM need to send SA path record queries to the SM to acquire the new addresses and path characteristics in order to reestablish the communication. Then, we apply our modifications that allows us to keep the same IB address after the migration, and we perform the same experiment. The cache is not enabled yet. We show that much less SA queries need to be sent to the SM in the connection reestablishment process. Last, we enable our novel caching mechanism and we migrate a VM again to demonstrate that peers can reestablish the connection without performing new SA path record lookups, since we know that the IB addresses will be the same after the migration.

B. Results

The RDS protocol and the *RDS-stress* test utility were used to demonstrate an application that communicates and tries to reconnect when the connection drops. The reason for the connection drop in our work is that one of the hosts participating in the communication is migrated, and the SR-IOV VF needs to be detached. RDS-stress has the notion of a server/client application only at the beginning of a test. The server is at the side waiting for the initial connection, while the client is the one initiating the connection. However, after the initial communication, there is no notion of server/client, as both sides actively send data. For the rest of the results, whenever server is mentioned, we refer to the node that waits for an incoming RDS-stress connection at the beginning of each experiment. We refer to the node that initiates the connection as the client. Three cases were studied:

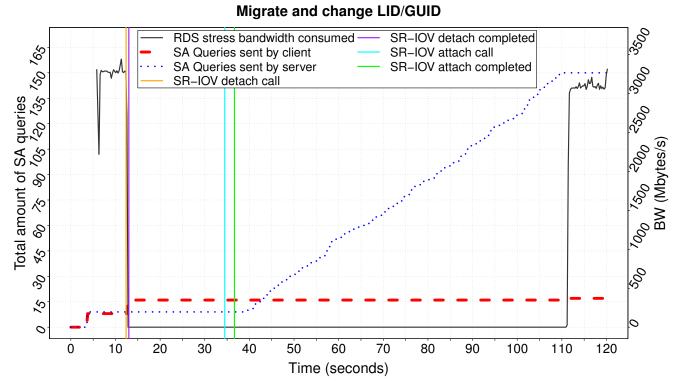


Fig. 5. Default behavior

1) Migrate a VM and do not keep the IB addresses: In Fig. 5 the server is migrated and the IB addresses change as part of the migration. The server was using LID_i and $vGUID_k$ when located at the source $hypervisor_y$, and got LID_j and $vGUID_m$ after moved to the destination $hypervisor_z$. This is the default behavior with the *Shared Port* SR-IOV model, and the worst case scenario as it is expected that participating nodes will send SA Path Record queries to the SM in order to find the new addresses and path characteristics to reconnect.

There are four vertical lines in Fig. 5. The first vertical line (orange) marks the moment the SR-IOV VF detach call was issued by OpenStack, right before the migration started. A device removal event is raised in the VM and the guest OS is freeing the resources using the device. The second vertical line (purple) marks the completion of the SR-IOV VF detachment. At this moment the SR-IOV VF was removed from the VM and the live migration started. The third vertical line (cyan) indicates that the live migration has finished, and the SR-IOV VF attachment procedure started at the destination hypervisor. The fourth line (green) marks the moment that the SR-IOV attachment has been completed from an OpenStack perspective. The guest OS needs some more time to load the drivers and reestablish communication. The solid measurement line shows the bandwidth consumed by the RDS-stress utility as measured at the client side. The thick dashed line and the thin dotted line show the number of the accumulated SA queries sent by the client and server sides respectively.

After four seconds, we see that both the client and the server send ~10 SA queries each, and two seconds later, the RDS-stress utility starts sending data. RDS-stress is responsible for this strange initial behavior, as RDS-stress first communicates by using TCP/IP to exchange the stress test parameters, then sleeps for two seconds, before both sides start sending traffic to maximize the throughput. Due to the fixed two seconds initial delay at both sides, both sides try to initiate a connection with the remote side simultaneously. Then, the RDS protocol experiences a race condition such as the one described in section IV-B2, Fig. 4, and the random backoff mechanism is used until the connection is established.

Around the 13th second, the SR-IOV VF detach is initiated at the server side that is going to be migrated. We see that at this moment, the client sends eight SA queries until the SR-IOV VF is detached at the server. These SA queries are sent by the client because the server is breaking the connection in order to detach the IB interface, and the underlying RDS protocol is trying to reconnect. However, until the IB interface is detached at the server, rejection messages are sent back to the client and the client retries. Then the migration starts and the client has an outstanding connection request that has not gotten a rejection back. At around the 39th second the migration has finished, the interface has been reattached and the drivers have been fully loaded at the server. Nevertheless, the RDS protocol on the server sends SA queries while trying to reconnect with the client, but all of the connection requests are rejected by the client. It is not until the 111th second —almost 70 seconds after the migration has been completed— that the connection is restored with a new connection establishment request from the client side. This behavior is observed because the client was trying to connect to the server at LID_i and $vGUID_k$ when the migration started. When the server came up again at the destination $hypervisor_z$ with a different LID_j and $vGUID_m$ and tried to connect back to the client, the client was rejecting the connection requests because the client already had an outstanding connection request towards LID_i and $vGUID_k$. At the 111th second, the client receives an *RDMA_CM_EVENT_UNREACHABLE* (timeout event) notification for its previous outstanding connection request and uses IPoIB to find the new GID address which is then the basis for the SA path record lookups. Eventually, RDS is using the GID address to send an SA Path Record query, gets the new path information and manages to establish the connection. Note that IPoIB uses ARP requests (via IB multicasts) to determine IP to GID mappings. If a re-ARP results in a new destination GID, or an IPoIB IP address becomes unreachable, then there is need for an additional SA path record lookup by the IPoIB protocol residing at all peer nodes.

It takes nearly 98 seconds to raise an unreachable event on an IB subnet with the default settings. The formula to calculate the response timeout as specified by the IB specs [7] is:

$$RESP = 4.096\mu sec \cdot (2 \cdot 2^{PLT} + 2^{RTV}) \quad (2)$$

In equation 2, where PLT = *Packet LifeTime* and RTV = *Response_Time_Value*. The subnet PLT value can be changed by the SM and the default value in *opensm.conf* is 18. The RTV value is hardcoded in the driver (*cma.c* source file) and its value is 20. Consequently, the timeout value is ~6.44 seconds as shown in equation 3.

$$RESP = 4.096 \cdot 10^{-6} sec \cdot (2 \cdot 2^{18} + 2^{20}) = 6.44 sec \quad (3)$$

There is also a hardcoded *Number of Retries* (NoR) value that is set to 15. The time needed before raising an unreachable event is $RESP \cdot NoR = 6.44sec \cdot 15 = 96.6sec$. As we can see, the observed time of 98 seconds that was needed for the connection to be reestablished is ruled by the default timeout values in the subnet. Moreover, in this prolonged period of inactivity, and because the server was operational at a much earlier stage before the actual connection was able to be reestablished, there was 140 excessive SA queries sent by the server while the server was trying to reconnect to the client. If more clients were connected to this server this number would be multiplied by the number of clients.

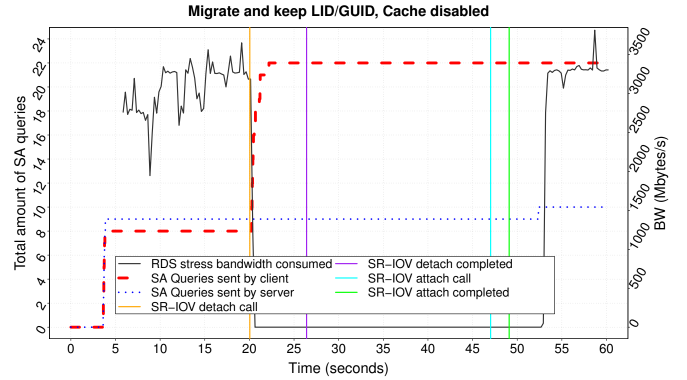


Fig. 6. LIDtracker enabled

2) Migrate a VM and keep the IB addresses with LIDtracker: In Fig. 6 the server is migrated together with the IB addresses by using LIDtracker. From the perspective of the migrated VM, the used addresses are LID_i and $vGUID_k$ when located at the source $hypervisor_y$ and the same LID_i and $vGUID_k$ are used after having moved to the destination $hypervisor_z$.

The initial RDS-stress behavior is as before (see V-B1). At around the 20th second the client sends 14 SA queries until the SR-IOV VF is detached at the server, but the situation is noticeably improved after the migration has been completed. In the previous case, when the LID and GUID changed at the destination, the client who had an outstanding connection establishment request waited for an unreachable event (~98 seconds) before accepting the next connection request. This behavior was inevitable because the client was trying to connect to the old address where the server is no longer present. In this case, where we keep the same destination address, the client does not have to wait for an unreachable event, as it can find the server even after the migration has been completed. Furthermore, the server is not flooding the network with SA queries; just a single one is needed to reestablish the connection.

3) Migrate a VM with LIDtracker and SA Path Record caching enabled: Although we reduced the amount of SA queries by keeping the IB addresses in section V-B2, there are still quite a few SA queries sent when the interface is detached, and at least one more is sent before the connection is reestablished after the migration has been completed. One additional observation is that the SA queries are sent in bursts when a disconnection event happens. For example, at the moment when the SR-IOV VF is detached from the VM that is going to be migrated, the remote nodes communicating with the *VM-to-be-migrated* will send a few SA queries each, in their attempt to reconnect as quickly as possible (explained in section V-B1). Consider the case that we migrate a VM that acts as a server, and that 2000 client nodes are communicating

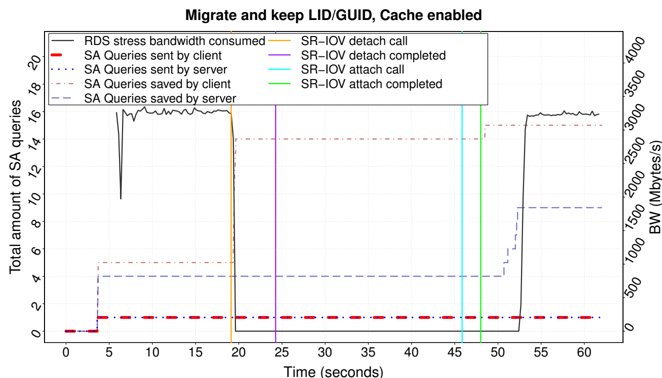


Fig. 7. LIDtracker and SA Path caching enabled

with it. If each of these clients sends an average of 10 SA queries in a single second when the migration of the server begins, the SM will be flooded with 20000 SA queries in a sub-second interval in addition to the normal load, which may be high already. As one can see, it is not necessary to have many concurrent live migrations in order to flood the SM with a significantly large load, but one migration of a popular node is enough. To alleviate the load towards the SM, we use the SA path caching mechanism introduced in section IV-B.

In Fig. 7 we demonstrate that connections can be made by using the local cache without sending any SA queries at all. Since we keep the IB addresses after the migration, the VMs can reconnect and communicate without introducing any additional load to the SM. We also introduce two more metrics in Fig. 7: *SA Queries saved by client* and *SA Queries saved by server*. These two metrics show the number of connection attempts from either the server or the client, that would have resulted in additional SA queries to be sent to the SM if our caching scheme was not enabled. As illustrated in Fig. 7, with our caching scheme enabled, only a single SA query is sent by the server and by the client. The connection attempts follow the trends presented in section V-B2, Fig. 6, but there are no subsequent SA queries sent to the SM after the initial one. The caching mechanism does not only reduce the SA queries sent as a result of live migrations, but also the total time it takes for one node to connect with another since there are less transactions. Looking back to the connection reestablishment procedure of RDS in Fig. 4, steps ④, ⑤, ⑨ and ⑩ are no longer needed, meaning that the connection reestablishment would complete quicker. If a similar caching mechanism was implemented in the drivers, applications in a single node would be able to share SA path records, resulting in much reduced overhead for the SM.

VI. Related Work

There is not much work done in the field of IB SA scalability in the context of cloud environments. Guay et al in [9] migrate VMs with SR-IOV VFs. They migrate the vGUID of the SR-IOV VF together with the VM, but the LID address changes. The main goal of their work is to reestablish the communication after a VM has been migrated and the LID address has changed, with the intention to reduce VM migration downtime. They implement a signaling mechanism based on the *repath* trap [7], to notify peers of the migrated VM to update their hardware address mappings, allowing the communication to be resumed. In this paper we take a different perspective and we show that no signaling or SM contact is required if we manage to migrate both the LID and GUID addresses together with the VM.

There is also an ongoing work in the OpenFabrics community, with the intention to create a distributed Scalable SA [18]. At the time of writing, there is not much information about this work other than a couple of presentations available online. Nevertheless, the Scalable SA project is focusing on providing a scalable implementation of the SM in order to solve the scalability challenges of the centralized nature of OpenSM, described in section III. Still, our work can be applied even in a more scalable implementation of the SM and reduce the load in the network. Furthermore, our work shows that a scalable SA path host caching mechanism can be even used in dynamic IB subnets where VMs can migrate and keep their associated addresses.

Other work related to the broader scope of scalable management of HPC network interconnects includes the path computation and distribution of routes on the switches. When an HPC network interconnet like IB grows in size, path computation and distribution on the switches is a time consuming process that ranges from a few seconds and can reach in the order of minutes[4] [22]. When faults occur, the network needs to be reconfigured and the path computation operation has to be repeated. The operational environment will perform suboptimally, or in the worst case suspended, until the new path computations complete and the reconfiguration applied. Gómez et al [23] propose a mechanism for distributed path computation to address the scalability issue of centralized path computation. Bermúdez et al [24], [25] use a set of suboptimal, but quickly calculated set of routes to bring the network in an operational state as quickly as possible. Then, the optimized set of routes is calculated offline and applied. Lysne and Duato [26] propose the *Skyline*, a method to identify the minimum part of the network affected by the fault and needs to be reconfigured. Bogdański et al [27] suggest segmentation of the network into manageable sections with the utilization of subnets, and propose two inter-subnet routing algorithms for IB.

VII. Conclusion and Future Work

In this paper, we presented evidence of the scalability challenges faced by the SM in an IB subnet. We also showed that VM live migrations contribute negatively to the scalability of the SM as the VM addresses change after the migration. Our novel SA Path Record caching mechanism was implemented in the RDS protocol, and the ability to (re)establish subsequent communication without sending SA Path Record queries was manifested — improving the scalability of the SM accordingly.

For now, and for the particular case of VM live migration, our prototype only supports IB subnets where the actual

[4]Network topology and chosen routing algorithm provide very diverse results.

path characteristics like service level and MTU, are the same before and after migration. To handle situations where live migration implies that the path characteristics may change (in heterogeneous subnets), is left for future work.

Our current prototype restarts the SM in order to migrate the IB addresses. However, the SM restart does not affect our experimental results because in our small cluster, the restart is quick and it is happening during the migration when the migrated VM has no IB network attached. Still, as future work we would like to identify how to migrate the IB addresses and update the Linear Forwarding Tables (LFTs) of the switches without restarting the SM.

Since we are using SR-IOV VFs, it is needed to detach/reattach the VF in order to live migrate. This process currently contributes significantly to the downtime of a migrating VM. Even with the late detach migration technique used by Guay et al. [17], the downtime is in the order of seconds. Also, highly active VMs with large amounts of memory will have longer downtime. In our experiments, we observed downtime variations in the range of 15 to 30 seconds for VMs with 4 GB of RAM. Here, downtime refers to the time needed for the SR-IOV VF to be detached, until the SR-IOV VF is reattached and the drivers are loaded after the migration has been completed. Clearly, more work needs to be done to reduce the downtime of live migration in combination with IB and SR-IOV.

To demonstrate our caching scheme, we used RDS. However, if a similar caching mechanism is implemented at the driver level, all of the applications running in a node could benefit from the cache. The caching scheme does not only reduce the load towards the SM, but also the connection latency between two applications since there is less communication overhead. If no path record request and response are needed because the local cache is used, less packets need to traverse the network before the actual connection attempt takes place.

ACKNOWLEDGEMENTS

We would like to acknowledge Mellanox Technologies for providing the IB hardware we used in our experiments. We would also like to thank Dr. Chao Jin who was the Shepherd of this paper, for his thorough review and valuable comments.

REFERENCES

[1] Simon Crosby and David Brown, "The virtualization reality," *Queue*, vol. 4, no. 10, pp. 34–41, 2006.

[2] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization.," *Intel Technology Journal*, vol. 10, no. 3, 2006.

[3] Patrick Kutch, "PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology," *Application note*, pp. 321211–002, 2011.

[4] Jithin Jose, Mingzhe Li, Xiaoyi Lu, Krishna Chaitanya Kandalla, Mark Daniel Arnold, and Dhabaleswar K Panda, "SR-IOV Support for Virtualization on InfiniBand Clusters: Early Experience," in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2013*. IEEE, 2013, pp. 385–392.

[5] Viktor Mauch, Marcel Kunze, and Marius Hillenbrand, "High performance cloud computing," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1408–1416, 2013.

[6] Marius Hillenbrand, Viktor Mauch, Jan Stoess, Konrad Miller, and Frank Bellosa, "Virtual InfiniBand clusters for HPC clouds," in *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*. ACM, 2012, p. 9.

[7] InfiniBand Trade Association, "InfiniBand Architecture Specification 1.2.1," 2007.

[8] TOP500, "Top500.org," http://www.top500.org/, 2014, [Online; accessed 03-March-2015].

[9] Wei Lin Guay, S.-A. Reinemo, B.D. Johnsen, T. Skeie, and O. Torudbakken, "A Scalable Signalling Mechanism for VM Migration with SR-IOV over Infiniband," in *IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), 2012.*, Dec 2012, pp. 384–391.

[10] "Reliable Datagram Socket," https://www.kernel.org/doc/Documentation/networking/rds.txt.

[11] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu,, and John Wiegert, "Intel® Virtualization Technology for Directed I/O," *Intel® Technology Journal*, vol. 10, no. 3, pp. 179—192, 2006.

[12] Carl Waldspurger and Mendel Rosenblum, "I/O Virtualization," *Communications of the ACM*, vol. 55, no. 1, pp. 66–73, 2012.

[13] Yaozu Dong, Xiaowei Yang, Jianhui Li, Guangdeng Liao, Kun Tian, and Haibing Guan, "High performance network virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471 – 1480, 2012, Communication Architectures for Scalable Systems.

[14] Gregory F Pfister, "An introduction to the InfiniBand architecture," *High Performance Mass Storage and Parallel I/O*, vol. 42, pp. 617–632, 2001.

[15] Jiuxing Liu, Wei Huang, Bülent Abali, and Dhabaleswar K Panda, "High Performance VMM-Bypass I/O in Virtual Machines.," in *USENIX Annual Technical Conference, General Track*, 2006, pp. 29–42.

[16] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[17] Wei Lin Guay, *Dynamic Reconfiguration in Interconnection Networks*, Ph.D. thesis, University of Oslo, 2014.

[18] Hal Rosenstock, "Update on Scalable SA Project," in *10th Annual OpenFabrics International Developer Workshop*, 2014, [Online; accessed 03-March-2015].

[19] Mellanox Technologies, "ConnectX®-3 Single/Dual-Port Adapter with VPI," http://www.mellanox.com/related-docs/user_manuals/ConnectX-3_VPI_Single_and_Dual_QSFP+_Port_Adapter_Card_User_Manual.pdf, 2014, [Online; accessed 03-March-2015].

[20] "Setting KVM processor affinities," https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Virtualization/ch33s08.html, [Online; accessed 03-March-2015].

[21] Mellanox Technologies Liran Liss, "Infiniband and RoCEE Virtualization with SR-IOV," in *2010 OFA International Workshop*, 2010, [Online; accessed 03-March-2015].

[22] Jens Domke, Torsten Hoefler, and Wolfgang E Nagel, "Deadlock-free oblivious routing for arbitrary topologies," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 616–627.

[23] Antonio Robles-Gómez, Aurelio Bermúdez, Rafael Casado, Åshild Grønstad Solheim, Thomas Sødring, and Tor Skeie, "A new distributed management mechanism for ASI based networks," *Computer Communications*, vol. 32, no. 2, pp. 294 – 304, 2009.

[24] A. Bermúdez, R. Casado, F.J. Quiles, and J. Duato, "Use of provisional routes to speed-up change assimilation in infiniband networks," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, April 2004, pp. 186–.

[25] A. Bermúdez, R. Casado, F.J. Quiles, and J. Duato, "Fast routing computation on infiniband networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 3, pp. 215–226, March 2006.

[26] Olav Lysne and José Duato, "Fast Dynamic Reconfiguration in Irregular Networks," in *Parallel Processing, 2000. Proceedings. 2000 International Conference on*. IEEE, 2000, pp. 449–458.

[27] Bartosz Bogdański, BjørnDag Johnsen, Sven-Arne Reinemo, and José Flich, "Making the network scalable: Inter-subnet routing in infiniband," in *Euro-Par 2013 Parallel Processing*, Felix Wolf, Bernd Mohr, and Dieter an Mey, Eds., vol. 8097 of *Lecture Notes in Computer Science*, pp. 685–698. Springer Berlin Heidelberg, 2013.