Realizing a Self-Adaptive Network Architecture for HPC Clouds

Feroz Zahid*†

Advisors: Ernst Gunnar Gran*, Tor Skeie*†

*Simula Research Laboratory, Norway †University of Oslo, Norway

***Abstract*—Clouds offer significant advantages over traditional cluster computing architectures including ease of deployment, rapid elasticity, and an economically attractive pay-as-you-go business model. However, the effectiveness of cloud computing for HPC systems still remains questionable. When clouds are deployed on lossless interconnection networks, challenges related to load-balancing, low-overhead virtualization, and performance isolation hinder full potential utilization of the underlying interconnect. In this work, we attack these challenges and propose a novel holistic framework of a self-adaptive IB subnet for HPC clouds. Our solution consists of a feedback control loop that effectively incorporate optimizations based on the multidimensional objective function using current resource configuration and provider-defined policies. We build our system using a bottom-up approach, starting by prototyping solutions tackling individual research challenges associated, and later combining our novel solutions into a working self-adaptive cloud prototype. All our results are demonstrated using state-of-the-art industry software to enable easy integration into running systems.**

I. Introduction and Motivation

Over the last decade, we have seen an incredible growth in the popularity of InfiniBand (IB) as a network interconnect for high-performance computing (HPC) systems and data centers. The recent *Top 500* supercomputer list, released in June 2016, reports that about 40.8% of the most powerful supercomputers in the world use IB as their interconnect. The popularity of IB is largely attributed to the high-throughput and low-latency communication it offers. More recently, the use of IB in cloud computing has also gained interest in the HPC community. Cloud systems built on an IB interconnect promises high potential of bringing applications which require a greater level of predictability and performance guarantees, such as HPC applications, to the clouds [1]. However, IB-based clouds that are designed oblivious to the underlying network topology and the installed routing algorithm, and without network optimizations based on the running workload fails to unfold the true potential. Challenges related to elastic load-balancing, efficient virtualization, and tenant performance isolation hinder full utilization of the underlying interconnect. On the other hand, the dynamic nature of the clouds, where tenant server machines are allocated, freed, and reallocated often, requires a self-optimizing network that takes the current resource configuration, network link loads, tenant node assignments, Service Level Agreements (SLAs), and provider-defined policies into account for optimization. Static network configurations normally used in HPC systems turns out sub-optimal and potentially insecure, requiring new rapid network reconfiguration schemes for IB to realize efficient HPC clouds.

In this work, we take on the challenge of designing a holistic self-adaptive framework for IB subnets realizing HPC clouds based on fat-trees, the most popular network topology for HPC systems. We build our system using a bottom-up approach, starting by prototyping solutions taking on individual research challenges associated with HPC clouds, with an proactive plan to combine them later into an integrated cloud prototype. A self-adaptive IB subnet will help achieving better network and system performance for the HPC clouds without any management interaction, running as an autonomous system. In addition, the resultant HPC cloud will improve QoS compliance and reduce SLA violations by proactive monitoring and optimization. More specifically, we address the following research challenges to realize an efficient cloud platform using IB systems and fat-tree topologies.

Challenge 1, Efficient Load Balancing: *What mechanisms are required to achieve efficient load-balancing on network links in the presence of distinct node traffic profiles in HPC systems.*

Challenge 2, Tenant Performance Isolation: *How to provide performance isolation to different tenants in a shared HPC cloud.*

Challenge 3, Fast Network Reconfiguration: *How to make network reconfiguration in IB fat-trees fast and compact.*

Challenge 4, Efficient Virtualization: *How to address scalability issues with virtual machine (VM) live migrations, and how to efficiently route virtualized IB subnets.*

Challenge 5, Self-Adaptive IB network for HPC clouds: *How to design and build a self-adaptive network architecture that can autonomously optimize itself according to the current resource configurations and provider defined policies.*

II. Our Approach and Results

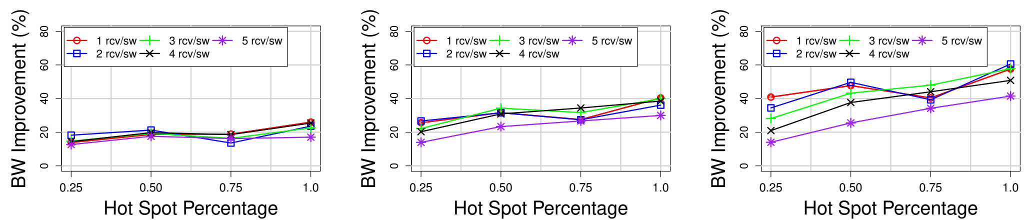
In this work, we follow the canonical action research methods [2] where the project is driven by the goal of building small working prototypes that meet the requirements identified in each of the challenges. All prototypes are demonstrated on a local test-bed using an IB-based cloud infrastructure. We use the OFED[1] software stack with OpenSM on top of Ubuntu to enable IB communication. For running MPI programs over RDMA, we use the MVAPICH2 MPI library. Several benchmarks are used throughout the work to evaluate our implementations including OFED's IB performance testing utility (*perftest*), the HPC Challenge Benchmark, the OSU Micro benchmarks, the Netgauge performance measurement toolkit, and the NAS parallel benchmark (NPB) suite. In addition, for large scale evaluation, we use simulations to complement the results we obtain from our test cluster. For flit-level simulations, we use an extended IB simulation model implemented in the OMNeT++ network simulation framework. We also use the *Oblivious Routing Congestion Simulator* (ORCS for simulating communication patterns on statically routed networks. Furthermore, we use OFED's *ibsim*, a tool that is distributed with the OFED software stack, to emulate physical topologies for generating routing tables using OpenSM.

We now present the solutions we have devised to address the challenges to realize a self-adaptive network architecture for IB-based HPC clouds.

A. Challenge 1: Efficient Load Balancing

For an efficient HPC cloud, it is highly important that the network links are balanced and network saturation is avoided. Network saturation can lead to low and unpredictable application performance, and a potential loss of profit for the cloud service providers. Furthermore, due to the dynamic workload admission, the network architecture should be able to reconfigure itself according to the current node traffic profiles. The current routing schemes used in IB fat-trees can be mainly categorized into either *deterministic*

[1]The OpenFabrics Enterprise Distribution (OFED) is the de facto standard software stack for building and deploying IB based applications. http://openfabrics.org/



(a) 18 switches with rcv nodes (b) 27 switches with rcv nodes (c) 36 switches with rcv nodes

Figure 1: Bandwidth improvement using wFatTree over fat-tree routing on a 648-node cluster.

or *adaptive*. Deterministic routing suffers from both static route assignment and rigid optimization assuming uniform traffic distribution, failing to adapt to changing traffic patterns and distinct node profiles. On the other hand adaptive routing, although promising higher degree of network utilization and load balancing, increases routing overhead, and may introduce out-of-order packet deliveries as well as degraded performance for window-based protocols.

In [3], we proposed a weighted fat-tree routing algorithm, wFatTree, which considers node traffic characteristics to efficiently balance load across the network links. The wFatTree routing is based on the notion of *weights* associated with each compute node. These weights are used to take known or learned traffic characteristics into account when calculating routes. The weight of a node reflects the degree of priority the flows towards a node receive when calculating routing tables. For example, a possible configuration could be to assign weights to the nodes in the range $[1, 100]$ depending on how much traffic a node is known to receive in the network. Such a scheme could assign $weight = 1$ for the nodes that receive very little traffic (primarily traffic generators, for example), and $weight = 100$ for the nodes receiving traffic near the link capacity. The values in between, $1 < x < 100$, will then reflect the proportion of traffic a node is expected to receive in the network. When no administrative information about the compute nodes is available, weights can be calculated using a simple port data counter based scheme. In OFED, a utility, *ibdatacounts*, is provided for reading data counters. After setting up the network with equal initial weights for all nodes, new weights can be *learned* after a specified time period. If B is the set of receive bandwidths for all the nodes measured over a time period, the weight for each node can be assigned in the range $[a, b]$ by using linear transformation as given by Equation 1.

$$W(x) = (x - a)\frac{b - a}{\max(B) - \min(B)} + a, \forall x \in B \qquad (1)$$

In Figure 1, the bandwidth improvements offered by wFatTree over the de facto fat-tree routing algorithm [4] commonly used in current IB systems are shown on a 648-port fat-tree with 36 leaf-switches. Receiver nodes (rcv) are the hotspot nodes in the network. As shown in the figures, as more traffic towards hotspots is generated or more leaf-switches are assigned hotspot nodes, wFatTree improves total network throughput by up to 60%, as compared to the fat-tree routing.

B. Challenge 2: Tenant Performance Isolation

Applications running on shared clouds are vulnerable to performance unpredictability and violations of the service level guarantees usually required for HPC applications. The performance unpredictability in a multi-tenant cloud computing system typically arises from server virtualization and network sharing. While the

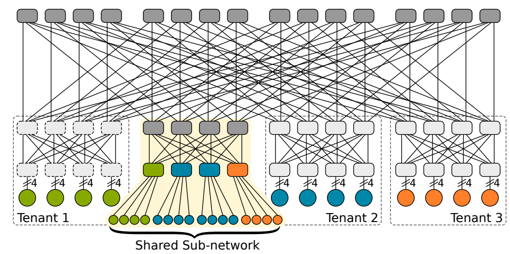


Figure 2: Tenant allocation in an example fat-tree network.

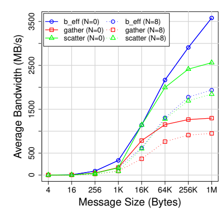
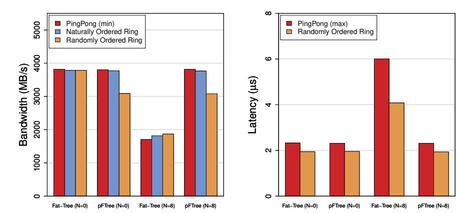


Figure 3: Effect of noise on average bandwidth for Fat-Tree routing.



(a) Bandwidth Tests (b) Latency Tests

Figure 4: The pFTree and Fat-Tree routing in noisy and noiseless cases (HPCC).

former can easily be addressed by allocating only a single tenant per physical machine, the sharing of network resources still remains a major performance variability issue. Intuitively, the network performance received by the applications of a tenant in a shared cloud is affected by the workload of other tenants in the system. The current IB implementation provides isolation mechanisms to enforce security through partitions, but does not provide those mechanisms at the routing level. This results in degraded load-balancing and interference among tenant clusters.

In hierarchical network topologies, like fat-trees, the tenants can be assigned to different leaf-switches or sub-networks providing network isolation inherited from the structure of the topology. However, such an allocation scheme only works for a restricted number of tenants, and for very rigid server requirements from each tenant workload, as shown in Figure 2. The switches that may need a change in routing for providing isolation are shown in dark gray color in the figure. Relying only on topology given isolation, only two of the three tenants can be supported leaving a plentiful of server machines unused. To tackle this problem, we presented partition-aware routing, pFTree, in [5], and later extended it to incorporate both provider-defined tenant-wise isolation policies and weighted load-balancing in [6].

The pFTree routing algorithm [5] aims to achieve two objectives in order of priority: first, it generates well-balanced linear

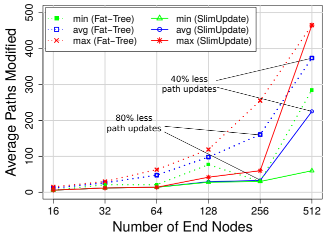
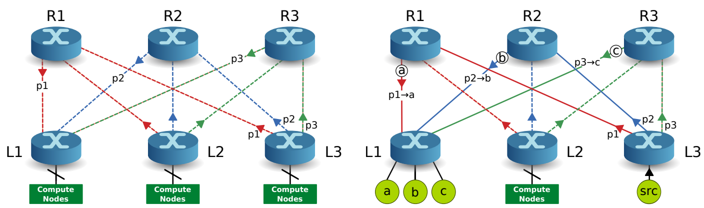


Figure 5: Total number of path modifications on reconfiguration.



(a) Phase I - Path Calculation (b) Phase II - Path Assignment

Figure 6: Two-phase leaf-switch based multipath routing.

forwarding tables (LFTs) for fat-tree topologies by distributing routes evenly across the links in the tree; second, while maintaining routes on the links balanced, pFTree removes contention between paths belonging to different partitions. The pFTree uses information about the subnet partitioning and ensures that the nodes in a partition receive a predictable network performance, unaffected by the workload running in other partitions. If the topology does not have enough links available to provide partition isolation (without compromising on the load-balancing), the pFTree assigns virtual lanes (VLs) to reduce the impact of contention between different partitions. The extended partition-aware routing, pFTree-Ext, presented in [6], additionally incorporate provider defined partition-wise policies that govern how the nodes in different partitions are allowed to share network resources with each other. These policies can be used to implement different SLAs for the tenants in an HPC cloud.

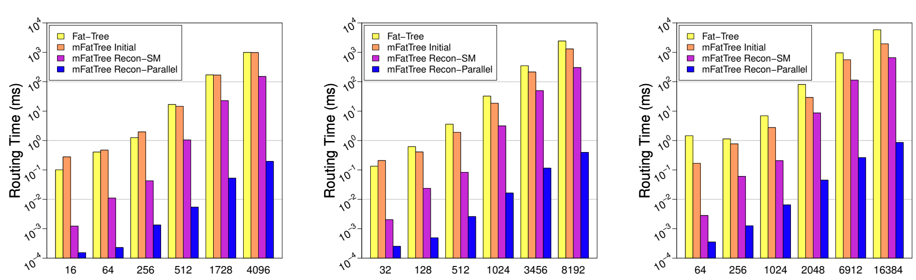
Figure 3 shows the impact of noise on the average bandwidth for different communication patterns when using the original fat-tree routing, as measured by Netgauge. The solid lines represent the measurements done with no noise from interfering partitions (N=0), while the dotted lines appear for the tests with the highest noise in our experiments (N=8). As shown in the figure, for all three communication patterns, we observe a substantial decrease in the achieved bandwidth. The pFTree routing algorithm in contrast substantially reduces the effect of noise on the victim partition, as noted by the bandwidth and latency benchmarks of HPCC.

C. Challenge 3: Fast Network Reconfiguration

The ability to efficiently reconfigure an interconnection network is an important feature that needs to be supported to ensure reliable network services for HPC clouds. In addition to handling faults, reconfiguration could also be needed to sustain network performance, and to satisfy runtime constraints defined by the provider policies. For instance, the routing function may need an update to optimize for a changed traffic pattern, or to maintain Quality-of-Service guarantees. In IB reconfiguration, the original routing function needs to be updated to cope with the change. The main shortcoming of the current reconfiguration techniques in IB is the costly re-routing for each reconfiguration event, making reconfigurations very expensive. To address those shortcomings, in [7] we presented a novel network reconfiguration technique, called Minimal Routing Update (MRU), and implemented SlimUpdate routing algorithm that uses MRU to ensure compact network reconfiguration.

The MRU and SlimUpdate Routing Algorithm: The MRU technique ensures that only a small subset of paths are updated during the reconfiguration, but still achieves the same routing performance as a full re-routing. The basic idea of the MRU technique is that, given a set of end nodes in a topology, multiple routing functions with the same performance and balancing characteristics can be generated. The SlimUpdate routing algorithm employs MRU technique to minimize total path modifications required on a reconfiguration event. As compared to the original fat-tree routing, SlimUpdate removes up to 80% ot the needed path modifications in most reconfiguration scenarios, as summarizes in Figure 5.

Metabase-aided Network Reconfiguration: To further optimize the IB reconfiguration mechanism, we presented a metabase-aided network reconfiguration scheme in [8] achieving fast reconfigurations for the fat-tree topologies. In the metabase-aided reconfiguration method, routing is divided into two distinct phases: calculation of paths in the topology, and assignment of the calculated paths to the actual destinations. For performance-driven reconfiguration, when reconfiguration is triggered without a topology change, the path calculation phase can be completely eliminated by using a metabase with stored paths from the first phase, hence saving routing time, which in turn substantially reduces overall network reconfiguration time. Moreover, once a set of calculated paths has been distributed to the switches, in principle, the re-routing phase can be executed in parallel at the switches further reducing time overhead. In addition, as the number of distinct paths is limited in a given fat-tree topology, the method reduces routing time for oversubscribed topologies too. Similarly, our metabase-aided network reconfiguration can also improve routing efficiency in virtualized subnets based on the vSwitch architecture [9], e.g. by enabling fast re-routing when a VM is migrated. In the first phase, the routing algorithm generates multiple paths between the leaf-



(a) Non-Oversubscribed Topologies (b) Oversubscription Ratio = 2 (c) Oversubscription Ratio = 4

Figure 7: Metabase-aided Routing significantly reduces the routing time for fat-tree topologies.

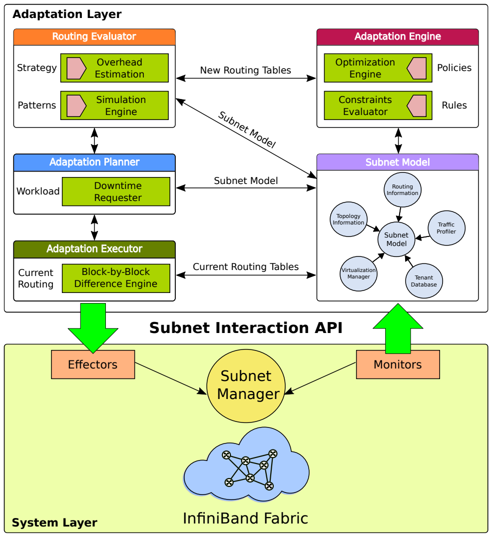


Figure 8: System Architecture

switches (or the switches to which end-nodes are connected), and the calculated paths can then be assigned to the actual destinations in the second phase. Two-phase leaf-switch based multipath routing on an example fat-tree network is shown in Figure 6. The routing times for metabase-aided routing, as compared to fat-tree routing are given in Figure 7. The initial routing times are referred to as mFatTree Initial, where mFatTree Recon-SM represents routing times when a metabase with calculated paths is used by mFatTree for generating new LFTs centrally using the subnet manager (SM). In addition, we show that, if the switches can be equipped with a parallel update having distributed the paths metabase, the reconfiguration time can be further reduced, shown as *mFatTree Recon-Parallel*.

D. Challenge 4: Efficient Virtualization

VMs is a salient feature of today's cloud systems. To meet the demands of HPC workloads, VMs need to utilize low overhead network communication paradigms, like Single-Root I/O Virtualization (SR-IOV). Yet, when passthrough-based virtualization is coupled with lossless interconnection networks, live-migrations introduce scalability challenges due to the substantial network reconfiguration overhead. The vSwitch IB architecture can be used to mitigate scalability issues, as described in [9]. We presented routing strategies for virtualized environments using vSwitches in [10].

III. The Big Picture: A Self-Adaptive IB Network

After implementing solutions for different challenges in the HPC clouds, we are now combining them all together in the form of a self-adaptive network architecture for HPC clouds. The architecture is inspired by the Rainbow framework [11]. The overview of our proposed system architecture for adaptive IB subnets is given in Figure 8. The architecture consists of two layers: a system layer and an adaptation layer. The system layer consists of the IB subnet manager (OpenSM), monitors, and effectors. The SM is responsible to configure and maintain an IB subnet. Monitors request the SM to get information about the subnet and its devices. Effectors define mechanisms to change the IB subnet, by installing a new set of routing tables, for example. The interaction between the system layer and the adaptation layer is defined by a *subnet interaction API*, implemented in the middleware.

The adaptation layer consists of five main components that interact with each other to provide self-adaptation capabilities to an IB network. A **Subnet Model** is maintained based on the information obtained from the monitors using the subnet interaction API. The subnet model includes information about the topology of the subnet, routing tables installed on the switches, and a database storing accounting information for the cloud tenants. The traffic profiles of the compute nodes are also maintained, using port data counter based monitoring service.

The core of the adaptation logic lies in the **Adaptation Engine**. The adaptation engine gets provider defined rules and policies as an input, and evaluates if the current subnet model satisfies the rules set by the provider, and/or if any further optimization is possible. The provider-defined policies are defined using an Event-Condition-Action based domain-specific language, implemented by means of bison parser generator. If an optimization is possible according to the rules set by the user, provider-defined policies and other configurable parameters are given as an input to a consolidated fat-tree routing algorithm. The consolidated fat-tree routing algorithm is implemented combining features of our implemented wFatTree, pFTree-Ext, and virtualized fat-tree routing. According to given parameters, the consolidated fat-tree routing algorithm calculates new routing tables. Once a new set of routing tables has been calculated by the adaptation engine, they are evaluated by the **Routing Evaluator** component of the adaptation layer. The routing evaluator estimates the overhead involved in the network reconfiguration required by the new set of routing tables. It also has an optional *simulation engine* where the new reconfiguration can be dry-run to find if there are enough potential benefits of installing the new configuration over the current one. The reconfiguration process itself is performed by the **Adaptation Executor**, which use a block-by-block difference engine to calculate the required blocks to be sent to the switches to install the new configuration.

IV. Conclusion

In this work, we address several challenges impeding realization of efficient HPC clouds based on IB interconnect. We present prototype solutions to achieve efficient load-balancing, tenant performance isolation, fast and compact network reconfiguration, and improved routing for virtualized environments. Furthermore, based on our prototype solutions, we present the design of a self-adaptive network architecture for IB subnets.

References

[1] A. Gupta and D. Milojicic, "Evaluation of hpc applications on cloud," in *Sixth Open Cirrus Summit (OCS), 2011*. IEEE, 2011, pp. 22–26.
[2] R. Davison, M. G. Martinsons, and N. Kock, "Principles of canonical action research," *Information systems journal*, vol. 14, no. 1, pp. 65–86, 2004.
[3] F. Zahid, E. G. Gran, B. Bogdański, B. D. Johnsen, and T. Skeie, "A Weighted Fat-Tree Routing Algorithm for Efficient Load-Balancing in InfiniBand Enterprise Clusters," in *23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2015.*, 2015, pp. 35–42.
[4] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized InfiniBand fat-tree routing for shift all-to-all communication patterns," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 217–231, 2010.
[5] F. Zahid, E. G. Gran, B. Bogdański, B. D. Johnsen, and T. Skeie, "Partition-Aware Routing to Improve Network Isolation in InfiniBand Based Multi-tenant Clusters," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2015.*, 2015, pp. 189–198.
[6] ——, "Efficient Network Isolation and Load Balancing in Multi-Tenant HPC Clusters," *Future Generation Computer Systems*, 2016.
[7] F. Zahid, E. G. Gran, B. Bogdanski, B. D. Johnsen, and T. Skeie, "SlimUpdate: Minimal Routing Update for Performance-Based Reconfigurations in Fat-Trees," in *1st HiPINEB Workshop, IEEE International Conference on Cluster Computing (CLUSTER), 2015*. IEEE, 2015, pp. 849–856.
[8] F. Zahid, E. G. Gran, B. Bogdański, B. D. Johnsen, T. Skeie, and E. Tasoulas, "Compact Network Reconfiguration in Fat-Trees," *The Journal of Supercomputing*, pp. 1–30, 2016.
[9] E. Tasoulas, E. G. Gran, B. D. Johnsen, K. Begnum, and T. Skeie, "Towards the InfiniBand SR-IOV vSwitch Architecture," in *2015 IEEE International Conference on Cluster Computing*. IEEE, 2015, pp. 371–380.
[10] E. Tasoulas, F. Zahid, E. G. Gran, K. Begnum, B. D. Johnsen, and T. Skeie, "Architectural Models and Methods for Efficient Virtualized Environments with Lossless Networks," *Submitted to IEEE Transactions on Network and Service Management*, 2016.
[11] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.