Network Optimization for High Performance Cloud Computing

Feroz Zahid

Doctoral Dissertation

Submitted to
the Faculty of Mathematics and Natural Sciences
at the University of Oslo for the degree of
Philosophiae Doctor

August, 2017

To Ahsan and Ifrah

Abstract

Once thought of as a technology restricted primarily to the scientific community, High-performance Computing (HPC) has now been established as an important value creation tool for the enterprises. Predominantly, the enterprise HPC is fueled by the needs for high-performance data analytics (HPDA) and large-scale machine learning – trades instrumental to business growth in today's competitive markets. Cloud computing, characterized by the paradigm of on-demand network access to computational resources, has great potential of bringing HPC capabilities to a broader audience. Clouds employing traditional *lossy* network technologies, however, at large, have not proved to be sufficient for HPC applications. Both the traditional HPC workloads and HPDA require high predictability, large bandwidths, and low latencies, features which combined are not readily available using *best-effort* cloud networks. On the other hand, *lossless* interconnection networks commonly deployed in HPC systems, lack the flexibility needed for dynamic cloud environments.

In this thesis, we identify and address research challenges that hinder the realization of an efficient HPC cloud computing platform, utilizing the InfiniBand interconnect as a demonstration technology. In particular, we address challenges related to efficient routing, load-balancing, low-overhead virtualization, performance isolation, and fast network reconfiguration, all to improve the utilization and flexibility of the underlying interconnect of an HPC cloud. In addition, we provide a framework to realize a self-adaptive network architecture for HPC clouds, offering dynamic and autonomic adaptation of the underlying interconnect according to varying traffic patterns, resource availability, workload distribution, and also in accordance with service provider defined policies. The work presented in this thesis helps bridging the performance gap between the cloud and traditional HPC infrastructures; the thesis provides practical solutions to enable an efficient, flexible, multi-tenant HPC network suitable for high-performance cloud computing.

Acknowledgments

I would like to thank my supervisors Ernst Gunnar Gran, Tor Skeie, and Olav Lysne for their continuous support, encouragement, and fantastic supervision throughout this doctoral work. In particular, I am grateful and honored to have a chance to work very closely with my principal supervisor Ernst during the course of this thesis (and after). I owe a lot to his guidance and I have always felt lucky to have him as my supervisor. Thank you Ernst! I am also very honored to have Tor as a mentor. Thank you Tor for selecting me for this PhD position, and also for your constructive feedback, insights, support, reviews, and for the many discussions we have had during my PhD years. Last but not least, I would like to express my gratitude to Olav for being a source of inspiration, and for believing in me and my work.

A big thanks goes to my friend and colleague Vangelis. It was very enjoyable to work, travel, and share office with you. We can rejoice that the neighboring offices did not complain too much about our numerous enthusiastic discussions, which had the inherent tendency to get louder as we went deeper in our conversations! I would also like to pay gratitude to our collaborators at Oracle Norway, Bjørn Dag Johnsen, Bartosz Bogdański, and Line Holen. In particular, Bjørn Dag and Bartosz have been a continuous source of ideas, constructive feedback, and reinforcement throughout this work.

Finally, I am very grateful to my family for the endless support I receive from them. In particular, my mother has always remained steadfast in praying for all of us. Thanks mom! A special thanks is due to my amazing wife Kehkashan – without her love, understanding, support, and continuous supply of tea and coffee, this work would never have been completed. Finally, lots of love is directed towards two wonderful bundles of happiness in my life, my son Ahsan and my daughter Ifrah. If it was not for their smiles, I would know no joy in life!

Table of Contents

List of Figures

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CBB	Cross Bisection-Bandwidth
CPU	Central Processing Unit
DLID	Destination Local Identifier
DNA	Deoxyribonucleic Acid
EDR	Enhanced Data Rate
FC	Fiber Channel
GbE	Gigabit Ethernet
GFT	Generalized Fat-Tree
GID	Global Identifier
GRH	Global Route Header
GUID	Globally Unique Identifier
HCA	Host Channel Adapter
HDR	High Data Rate
HPC	High-performance Computing
HPDA	High-Performance Data Analytics
I/O	Input/Output

IBA	InfiniBand Architecture
IBTA	The InfiniBand Trade Association
IB	InfiniBand
ICT	Information and Communication Technology
IoT	Internet-of-Things
IPC	Interprocess Communication
iWARP	internet Wide Area RDMA Protocol
LAN	Local Area Network
LFT	Linear Forwarding Table
LID	Local Identifier
MIN	Multistage Interconnection Network
MPI	Message Passing Interface
NIC	Network Interface Card
OFED	OpenFabrics Enterprise Distribution
QFT	Quasi Fat-Tree
QoS	Quality-of-Service
QP	Queue Pair
RDMA	Remote Direct Memory Access
RLFT	Real Life Fat-Tree
RoCE	RDMA over Converged Ethernet
RUFT	Reduced Unidirectional Fat-Tree
SCSI	Small Computer System Interface
SLA	Service Level Agreement

SLO	Service Level Objective
SL	Service Level
SMA	Subnet Management Agents
SMI	Subnet Management Interface
SMP	Subnet Management Packet
SM	Subnet Manager
TCA	Target Channel Adapter
VL	Virtual Lane
VM	Virtual Machine
XGFT	eXtended Generalized Fat-Tree

Part I

Overview

Chapter 1

Introduction

Since the inception of the early supercomputers in the 1980s, the high-performance computing (HPC) industry has strived for a continuous growth in performance. This can be seen by the trends on the Top 500 supercomputer list [1], which show a near exponential increase in computational power over the last twenty years. Traditionally, the computational power provided by HPC installations has been mainly used by the scientific community. From untangling complicated sequences in the DNA to simulating intricate meteorological phenomena, HPC has proved to be crucial in solving complex problems that require very high compute and network performance, unavailable on conventional computing platforms [2–5]. However, over the last few years, a growing interest in high-performance data analytics (HPDA) and machine learning at scale have fueled the need of HPC for the enterprises [6,7]. In areas as diverse as manufacturing, financial services, digital media, business intelligence, information security, and Internet-of-Things (IoT), organizations create or collect very large datasets, often at speeds surpassing what we can handle using traditional data management techniques. Within the huge amount of data produced, commonly referred to as *big data* [8], lies a great potential in the form of undiscovered structures and relations [9]. To realize this potential, gain new knowledge, and create business value, the data produced needs to be accessed, processed, analyzed, and visualized – in the most efficient manner. This is where HPC comes into play [10–12].

Cloud computing [13–16], a new computing paradigm characterized by elasticity and on-demand network access to computational resources located at a *cloud data center*, offers a momentous potential of bringing HPC capabilities to a broader enterprise audience. However, clouds employing traditional *lossy* network interconnects, such as commodity Ethernet [17], are generally not very successful when it comes to HPC [18]. Both the traditional HPC workloads and HPDA require high predictability, large bandwidths, and low latencies, features which are, in general, not readily available using *best-effort* cloud networks [19]. On the other hand, *lossless*[1] interconnection networks commonly deployed in HPC systems, lack the flexibility needed for cloud environments. For instance, the dynamic nature of the cloud owing to multi-tenancy, rapid elasticity and on-demand resource provisioning, renders static network configurations commonly employed in HPC interconnects infeasible or sub-optimal.

This thesis identifies and addresses research challenges that hinder realization of an

[1]Contrary to the lossy networks, *lossless* interconnects do not drop packets in normal network operations by using a flow-control mechanism.

efficient HPC cloud computing platform, utilizing a lossless interconnect technology, like InfiniBand [20]. The work presented in this thesis bridges the performance gap between clouds and traditional HPC infrastructures, by providing practical solutions to enable an efficient, flexible, multi-tenant HPC network architecture suitable for the clouds.

In this chapter, we provide the motivation and context of this work. We present the problem statements and research questions of the thesis, summarize research methods employed, and outline the structure of the rest of the thesis.

1.1 Motivation

The amount of digital data in our world has become enormous and is growing at exponential rates. It is estimated that the size of the *digital universe* will grow from 4.4 zettabytes[2] in 2013 to 44 zettabytes by 2020 [21]. A major factor influencing this rapid data explosion is the growing popularity of IoT. IDC [22], a global marking intelligence firm, forecasts that the installed number of IoT devices is expected to grow from 15.4 billion devices in 2015 to 30.7 billion devices in 2020, and the growth is expected to continue at a rate of 20% per year in the next five years [23]. These billions of devices have sensors and other kinds of data generators producing enormous amount of data awaiting processing and analysis for value creation. Formally, big data refers to the datasets whose size is beyond the ability of conventional software tools, like relational databases [24–26], to store, manage and analyze. However, size and *volume* are not the only features that prescribe which data to be classified as big data and which to be not. Besides plausibly the high volumes of data, big data also imposes challenges of handling the *variety* of these volumes with different forms (structured and unstructured), at a considerably high *velocity* or transfer rate. The volume, velocity, and variety, together make up the three most important challenges associated with the management of big data in communication networks, and are referred to as the three *Vs* in the literature [27]. Besides these three Vs, it is equally important that the *value* can be extracted, even in the presence of *veracity* or uncertainties in the collected data [28]. An important point to note here is the fact that as more and more digital data is being produced in different areas, many of the computational problems formerly known to be associated with structured or low volume data, for instance data query, are converging to big data problems – pushing the need for efficient big data processing and management [29].

The Need for an HPC Cloud

The idea that computational power could be provided to the users as a *utility* is not new and can be dated back to 1966 [30]. However, it was not until recently that the term ‘cloud’ started gaining popularity, referring to solutions offering a broad array of computing and storage services over the Internet [31]. Over the last few years, cloud computing has principally caused a paradigm shift in computing, and the industry has witnessed an accelerated transition from small-scale, closed computing and data storage architectures to large, open and service oriented infrastructures [32]. Cloud architectures offer significant advantages over traditional cluster computing architectures including flexibility, ease of setup

[2] 1 zettabyte = 10^{21} bytes

and deployment, high-availability, and on-demand resource allocation - all packed up in an economically attractive pay-as-you-go [33] business model for its users.

Traditionally, HPC resources were almost exclusively deployed and committed by large research institutes, universities, national laboratories, and governmental bodies. The engineers and scientists, being HPC users, normally had to wait long before getting access to the highly sought-after HPC resources for their applications. With the emergence of big data workloads as the new HPC *killer application*[3] arises the need for extending HPC resources to a much wider audience in a flexible and cost-effective way. For example, the European Technology Platform for High-Performance Computing (ETP4HPC) in its 2015 update to the technology multi-annual roadmap towards Exascale[4] computing, highlights that *besides traditional HPC workloads, more and more big data applications will need to be addressed with HPC solutions* [34]. The report also notes that the cloud computing delivery model will impact the features of the future HPC systems. A good example of cloud usage for the traditional HPC can be taken from a recently published news report [35] about *HyperXite*, which is a company that focuses on building future transport systems, such as *Hyperloop* [36]. Competing in a *SpaceX*-sponsored Hyperloop competition, a student team had to run highly computationally demanding simulations for fluid dynamic modeling to find ways to reduce drag, minimize mass, and maximize speed for the Hyperloop. Such a workload is a traditional HPC workload requiring a supercomputer to run on. However, available private HPC infrastructure was not feasible as the time required to run those simulations was simply too long given the available resources. The team used the Microsoft Azure [37] public cloud infrastructure to compute results for their modeling experiments. This shows the potential clouds have even for traditional HPC applications.

Arguably, through HPC clouds, a large number of enterprises, as well as research institutes and academic organizations, could benefit from feature-rich cloud offerings, potentially saving them substantial capital expenditure while providing *instant* and *elastic* resource capacity for their applications. However, in practice, effective use of cloud computing for HPC systems still remains questionable. Applications running on shared cloud networks are vulnerable to performance unpredictability and violations of service level agreements [38–42]. On the contrary, HPC applications typically require predictable network performance from the infrastructure. This shortcoming of shared clouds is also reflected in the market uptake of cloud computing for HPC workloads. A recent market study published by Intersect360 Research [43], despite mentioning machine learning as a key new trend, shows a lack of market growth for HPC in the public clouds [44]. The report suggests that the market remains selective with respect to the jobs it offloads to the cloud platforms.

The work presented in this thesis is motivated by the needs of future data centers aiming to provide efficient HPC cloud solutions to increase cloud uptake for both big data and traditional HPC applications.

[3]Coined by PC Week in 1987, the term ‘killer application’ is used to refer to a software application so important for customers that it drives popularity of some larger technology, such as computer hardware or platform.

[4]Exascale implies computational power capable of performing 10^{18} double-precision floating-point operations per second (exaFLOPS).

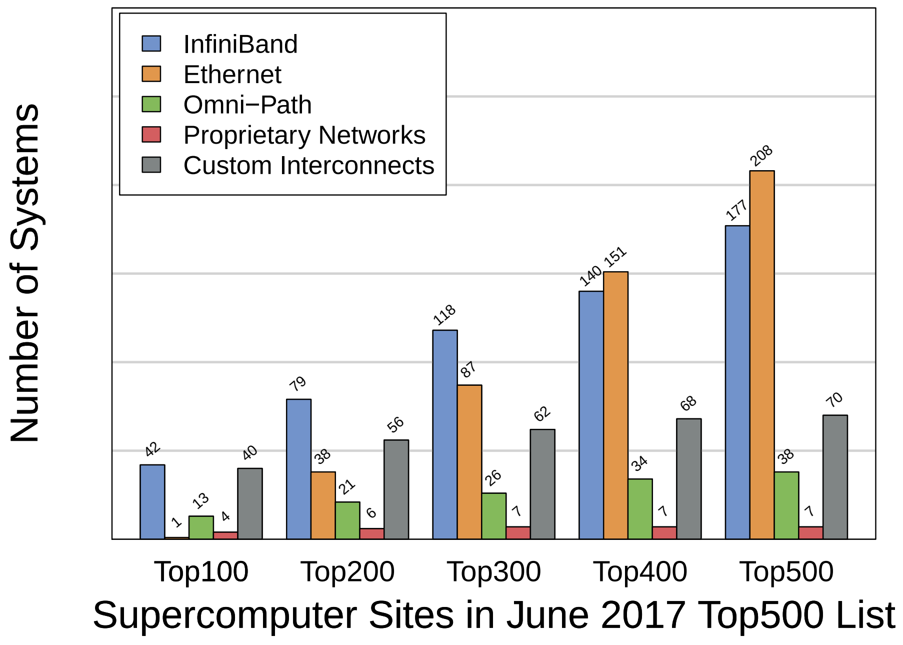


Figure 1.1: Interconnect technology share in Top500 Supercomputers

InfiniBand as a Cloud Interconnect

A scalable and efficient data center network is essential for a performance capable cloud computing infrastructure. HPC and HPDA applications, in particular, demand high-throughput network connectivity and low latency communication due to the abundant use of parallelization. Applications, such as those used in large-scale simulations, big data analytics, and machine learning, require frequent, irregular, and data-intensive communication between processing nodes, making the network an important determinant of the overall application performance [45,46]. Moreover, studies conducted on public Infrastructure-as-a-Service (IaaS) [47] cloud offerings like Amazon EC2 [48], identify the network as a major performance bottleneck for efficient execution of HPC applications in the cloud [40–42,49,50]. The performance disparity between HPC systems and cloud infrastructures is chiefly because many current cloud systems use low-cost commodity Ethernet networks providing relatively low bandwidth and high latency between nodes. HPC interconnect technologies [20,51–55], on the other hand, use specialized hardware to provide robust high-throughput low-latency network interconnects in HPC installations.

InfiniBand (IB) [20] is an open standard lossless network technology developed by the IB Trade Association (IBTA) [56]. Over the last decade, we have seen an incredible growth in the popularity of IB as a network interconnect for HPC systems and data centers. The recent *Top500* supercomputer list [1], released in June 2017, reports that more than 35% of the most powerful supercomputers in the world use IB as their interconnect. As shown in Figure 1.1, 177 of the Top500 supercomputer sites use IB interconnect, surpassed only by the Ethernet family including Gigabit and 10-Gigabit Ethernet (GbE, 10GE). However, IB has a much larger share than Ethernet with about 30% more number of systems in the *Top300*, as shown in the figure, suggesting that the performance characteristics of IB makes IB more suitable for very large supercomputer installations.

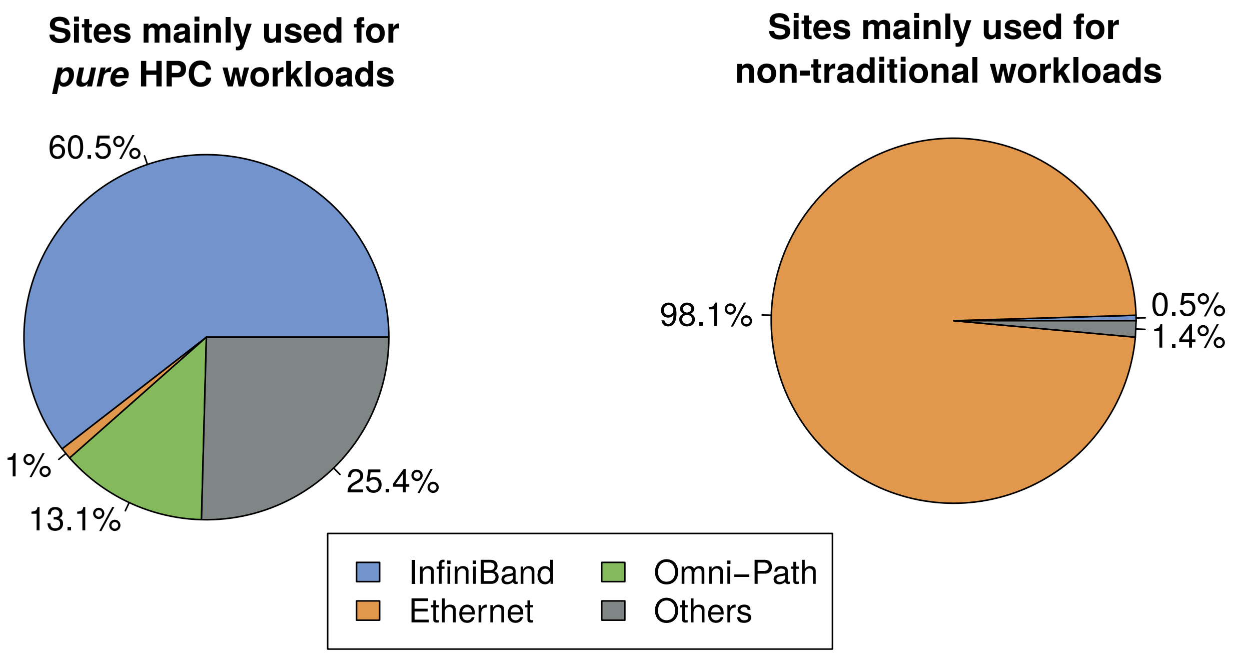


Figure 1.2: In Top500 list, InfiniBand is dominant among HPC systems, and almost non-existent on sites that are mainly used for nontraditional and emerging workloads, such as clouds.

Recently, the use of IB in cloud computing has also gained interest in the HPC community [57–61]. Thanks to the high-throughput and low-latency communication offered by IB, cloud systems built on top of an IB interconnect promise high potential of bringing HPC and other performance-demanding applications to the cloud. Furthermore, IB provides sufficient security mechanisms to complement in typical non-trusted data center environments. However, when clouds are deployed on IB interconnects, challenges related to load-balancing, low-overhead virtualization, efficient network reconfiguration, performance isolation, and dynamic self-adaptation obstruct full potential utilization of the underlying interconnect. This is mainly due to the lack of flexibility stemming from the very fundamentals of how IB works. To gain performance in IB, most of the communication-related work is offloaded to the hardware. Moreover, for the same reason, routes are generally static based on *linear forwarding tables* (LFTs) stored in the switches. Contrary to the traditional HPC systems, clouds exhibit a very dynamic environment, where new tenant machines are allocated, migrated, freed, and re-allocated often. The non-flexibility of IB to quickly adhere to varying configurations make IB less suitable for the clouds, and other non-traditional-HPC workloads. Even though some cloud providers, for instance Microsoft Azure [37], OrionVM [62], and ProfitBricks [63], provides compute instances connected using an IB interconnect (mainly through dedicated hardware resources), the aforementioned challenges still needs to be addressed to enable IB for a broader cloud adaptation.

The suitability of IB for the traditional HPC installations, and lack of popularity with other installations such as clouds, can be seen from having a closer look at the Top500 list, as done in Figure 1.2. As mentioned previously, supercomputers are increasingly being used to run non-traditional HPC workloads, such as machine learning applications [64]. At the same time, performance improvements on very large non-HPC installations such as Web2.0 [65] *hyper-scale* platforms [66] and clouds, have resulted in the inclusion of many sites primarily used for the non-traditional emerging HPC and non-HPC workloads, in the Top500 list.

Studies [67, 68] show that the current Top500 list contains just about 290 systems that are used for *real* or traditional HPC purposes[5]. Among these systems, about 60% of the sites use IB. On the other hand, IB is almost non-existent when it comes to the sites that are not used for traditional HPC. Ethernet dominates those systems with more than 98% share owing largely to the flexibility Ethernet offers.

In this work, we propose solutions to better adapt IB for HPC cloud computing and make it suitable to run non-traditional and emerging HPC workloads, as well as traditional HPC applications, in the cloud.

Routing is a Key Determinant in Interconnection Networks

Routing plays a crucial role in HPC systems, and optimized routing strategies are required to achieve and maintain optimal throughput and low latencies between nodes [69]. For large-scale HPC systems with multiple tiers of compute nodes, as expected from an HPC cloud, it is important to exploit the physical topology and the availability of path diversity between nodes to achieve optimal network performance. *Spanning tree* based protocols [70], typically employed in commodity networks to avoid loops, are unable to exploit the topology characteristics, for example in *fat-trees* [71]. On the other hand, in HPC interconnection networks, a plethora of work is dedicated to improving routing performance. A large number of topology-specific routing algorithms [72–74], as well as topology-agnostic routing algorithms [75–78] have been proposed. However, many of the algorithms are typically designed for improving routing performance for HPC workloads, often optimizing for all-to-all communication patterns. Some application specific routing algorithms have also been proposed [79–81], but those too typically address routing in a non-dynamic HPC setting, contrary to the one required for the dynamic cloud networks. Typical challenges in a cloud environment, such as elastic load-balancing, efficient virtualization, and tenant performance isolation, can only be addressed in an IB system when routing is done making use of cloud specific information, for example, location of the tenant nodes, virtualization information, and node roles, into consideration. In addition to that, due to the dynamic nature of the clouds, fast and autonomic network reconfiguration solutions are important to keep the network optimized according to changing HPC workloads, requirements, and tenant allocations. Furthermore, cloud system optimization criteria themselves are also provider-specific. For example, during hours of low load, a cloud provider may like to save power by using server consolidation and shutting down unused machines, while during peak hours, the same provider may like to distribute load across the data center evenly to combat network congestion. This implies that to achieve optimizations as required by the cloud service provider, the optimization strategies set by the provider must also be taken into account. The aforementioned shortcomings of the state-of-the-art IB solutions when it comes to the support for dynamic environments, motivate the doctoral work presented in this thesis.

The main focus of this thesis is on IB networks built using the fat-tree topologies [71]. The fat-trees, first introduced by Leiserson [71] in 1985, are a class of general-purpose network

[5]While the distinction between HPC and non-HPC platforms (and workloads) is purely hypothetical, it is a very useful distinction to comprehend the evolving HPC ecosystem. We define HPC platforms as the ones running traditional HPC applications comprising mainly modeling and simulations; while non-traditional workloads are assumed to include machine learning, artificial intelligence (AI), and big data analytics, among others.

topologies shown to scale with the availability of the network resources. Fat-trees, together with its variants, make a very popular network topology for HPC systems and data centers. For example, the current second fastest supercomputer in the Top500 list [1], Tianhe-2 (or Milky Way-2 after its Chinese name) [82], at the National Supercomputer Center in Guangzhou uses a fat-tree topology. Tianhe-2 remained the fastest supercomputer in the world from June 2013 until June 2016 when Sunway TaihuLight [83] surpassed Tianhe-2 to become the world's fastest supercomputer. A detailed background on the fat-tree topologies is provided in Section 2.2.1.3.

The *OpenSM*, which is the open-source subnet manager for the IB bundled with the OFED sofware stack[6] [84] supports several routing algorithms [76–78, 85]. We have particularly focused on proposing improvements to the fat-tree routing algorithm [86, 87] to make it better suitable for dynamic HPC clouds. However, most of the concept presented in this thesis can be applied to other routing algorithms, topologies, and interconnection network technologies as well.

1.2 Research Questions

The interconnection network is an important resource affecting the overall efficiency, cost, and performance of a networked system, such as a supercomputer or an HPC cloud [69, 88]. It is important that the available network resources are utilized in the most efficient manner, specifically through careful allocation of routes to the available network links. Underutilized network resources decrease the overall efficiency of the system and result in higher costs and lower application performance.

Most current routing algorithms, such as the de-facto fat-tree routing algorithm [87] implemented in OpenSM, spreads routes across the links in the network by balancing the number of routes assigned to each link equally. The main problem with this load-balancing technique is that it assumes a uniform traffic distribution in the network. However, end nodes[7] in an HPC cloud, and generally in any multi-tenant cluster, are not sending and receiving traffic in a uniform way. Different nodes are subject to different requirements, exhibit varying traffic characteristics, and need to adhere to distinct roles in the system. When treated equally by the routing algorithm, all nodes get the same priority, and hence, depending on the routing technique used, ideally equal or similar network resources are assigned to them. Such node oblivious routing results in sub-optimal network utilization and improper load-balancing. For instance, when routes towards nodes that mainly consume large amount of data are assigned to share links in the fabric while alternative links are underutilized, the situation results in an overall sub-optimal network throughput. In particular, for enterprise HPC clusters, some nodes have pre-assigned roles determining their traffic profiles and requirements. That is, storage nodes or I/O gateways are generally expected to receive a large amount of the total network traffic and can be termed as network hot spots *a priori*. Hence, the links towards those nodes are more likely to be congested and need priority balancing to achieve optimal overall network throughput in the cluster. Similarly, flows towards some

[6]The Open Fabric Enterprise Disribution (OFED) is the de facto standard software stack for building and deploying IB based applications. http://openfabrics.org/

[7]We use the term *end nodes* or simply *nodes* to refer to both compute and storage nodes in this thesis.

critical nodes may also need high priority treatment in the network. The same is true for the HPC clouds, where different tenant may have different *service level agreements* (SLAs) with the service provider [89], determining the service their nodes are expected to receive in the network. Tenant-oblivious routing in that case will fail to distinguish between the SLA requirements from various tenants and will treat their traffic in the network irrespective of the SLAs.

While load balancing in general is a recognized research problem in cloud computing, the majority of the existing solutions conduct load balancing based on mapping between the requirements and the available resources in the cloud [90–93]. However, in case of an HPC system, as mentioned above, the routing algorithm determines how the load in the network is balanced across the available network links. On this end, *adaptive routing* [94] can be used for load-balancing, as employed by several load balancing algorithms in interconnection networks [95–97]. Adaptive Routing, characterized by its ability to adhere to traffic conditions such as congestion in the network, although promises higher degree of network utilization and load balancing, increases routing overhead, and might introduce out-of-order packet deliveries, as well as degraded performance for window-based protocols [98]. Another important issue with current load-balancing techniques is the lack of predictable network performance, in particular, in dynamic environments such as a cloud. Node-oblivious route assignment results in different performance output when nodes, as well as their roles, are changing relatively fast. Thus, our first research question is:

> *RQ1: What are the implications of node-oblivious routing in HPC cloud systems, and what mechanisms are needed to achieve high network utilization and predictability in HPC networks like fat-trees?*

Efficient load-balancing techniques can improve network utilization, and thus, the performance and predictability of an HPC cloud. However, predictability is also challenged by the very nature of cloud computing. By definition, clouds provide a shared resource model where multiple tenants are served from the same data center infrastructure. The sharing of resources may result in unpredictable application performance [40, 79, 99–101]. The performance unpredictability in a multi-tenant cloud computing system typically arises from server virtualization and network sharing. While the former can be addressed by allocating a single tenant per physical machine, as employed by major HPC cloud providers like Amazon in their HPC offerings, the shared network infrastructure still remains an issue. From the networking perspective, ideally each tenant should experience predictable network performance, unaffected by the workload of other tenants in the system. A naïve approach could be to dedicate network resources to each tenant *sub-system*. However, this may lead to underutilization of the available network resources as different tenants may have different requirements from the interconnect. Dedicated network resources for a tenant machine with only limited communication to other machines in the system, or for a tenant cluster that is confined to a single switch, for example, may lead to waste of network resources. This situation, understandably, is contradictory to the requirements mentioned in the description of our first research question.

Network isolation mechanisms are not sufficiently implemented in IB systems to efficiently support cloud environments. Even though IB provides *partitioning* and Quality-of-Service (QoS) mechanisms to ensure isolation between nodes in the network, the problem with the current routing algorithms is that they do not take partitioning information into consideration when assigning routes in the network. This situation leads to both degraded load-balancing and lack of performance isolation among tenant nodes. We are now ready to present our second research question.

> *RQ2: What is the impact of network sharing on application performance in a shared-network infrastructure, and how can we provide performance isolation to the tenants in a shared HPC cloud while keeping the network utilization high?*

Routing optimizations based on efficient load-balancing and network isolation are general optimization criteria for HPC clouds. However, for large and dynamic HPC systems in practice, the optimization problem becomes both complex and multi-dimensional, while individually proposed solutions in the absence of a global *utility* or objective function, often yield contradictory management decisions. In addition, node traffic profiles and tenant group information form just two of many criteria a service provider may need to consider, determining the routing of the network. Other criteria, such as system-wide goals, energy requirements, costs, SLAs and service level objectives (SLOs), can be just as important for a service provider. A feature completely missing in the current IB stack is the provision of *tuning in* the routing algorithm according to the provider-defined policies, requirements, and constraints. Routing algorithms work on general optimization criteria and are unable to adhere to the requirements of the service provider, thus, providing low utility, as perceived by the provider. This missing feature of the routing algorithms lead us to the next research question:

> *RQ3: How can we incorporate service provider-defined policies and system-wide goals in the routing algorithms for HPC systems?*

Virtualization [102] is an important feature providing flexibility, fast deployments, and fault-tolerance in cloud computing. To meet the demands of communication-intensive workloads in the cloud, virtual machines (VMs) employ pass-through techniques, like *Single Root I/O Virtualization* [103], to be able to directly communicate with Input/Output (I/O) hardware and reduce overhead. Due to the dynamic nature of the cloud, VMs need to migrate among physical machines for fault-tolerance, power-saving, or to mitigate server *fragmentation* [46]. However, with SR-IOV-based virtualization employed on IB, VM *live migrations* introduce scalability challenges and substantial network reconfiguration overhead, largely due to the rigid IB addressing schemes (IB addressing is detailed in Section 2.4.1). A *virtual switch* (vSwitch) SR-IOV architecture [104] can be used to mitigate this overhead. The vSwitch architecture provides a complete set of IB addresses to VMs residing on the same physical machine contrary to the *shared-port* model where VMs share the addresses of the physical machine[8]. From the routing perspective, for example in fat-tree routing, VMs can be routed independently of other VMs attached to the shared vSwitch as the VMs have individual addresses. However, the single upward link from the vSwitch to the corresponding physical switch remains the bottleneck. Thus, our fourth research question is:

[8]A more modular virtualization approach has since been implemented in IB, based on the concept of *vPorts*. Interested readers are referred to [105] and [106] for further details.

> *RQ4: What are the requirements of efficient virtualization in lossless interconnection networks, and how can we efficiently route virtualized HPC topologies?*

The last two research questions addressed in this thesis stem from the fact that, as mentioned previously in this chapter, cloud environments are dynamic. In large HPC clouds, the number of events requiring a network reconfiguration, as well as the complexity of each reconfiguration, is likely to increase with growing system sizes [107]. These events include component failures, node additions/removals, link errors etc. In addition to handling faults, reconfiguration is also needed to maintain or improve network performance, and to satisfy runtime constraints, such as those defined by the service provider. For instance, the routing function may need an update to optimize for a changed traffic pattern, or to maintain QoS guarantees for a tenant. Similarly, modern energy-saving techniques rely on server consolidation, VM migrations, and component shutdowns to save power [108]. In all these events, the original routing function needs to be updated to cope with the changes. Dynamic network reconfiguration in statically routed IB networks requires computation and distribution of a new set of routes to the switches, which introduces substantial overhead. In addition, manually-triggered routing updates are inefficient, due to the dynamics of the cloud, dynamics that often require frequent network reconfigurations based on current cloud configurations, monitored network conditions, and live application metrics. The last two research questions, *RQ5* and *RQ6*, are:

> *RQ5: How can we minimize network reconfiguration time and cost in HPC systems?*

> *RQ6: What are the requirements of a self-adaptive network architecture, employed using lossless interconnection networks like IB, that can autonomously optimize itself according to the current resource configurations, traffic conditions, and any provider-defined policies and constraints?*

In the next section, we present the research methods employed throughout the work of this thesis to answer the research questions presented in this section.

1.3 Research Methods

The primary goal of research is to produce new knowledge or gain a deeper understanding of a topic [109]. A research method defines the specific approach and systematic procedures used in *search* of this new knowledge or understanding. The research methods commonly employed in the field of computer science can be broadly classified into two categories based on the research paradigm used: *Analytical approaches* and *Experimental approaches* [110]. The analytical or theoretical approaches are based on formal theories, where studied systems are first mapped to mathematical or statistical models, and then analyzed using existing tools of mathematics and logic to gain new knowledge about the subject [111]. The experimental approaches, on the other hand, rely on the evaluation of existing or proposed concepts and solutions through design of experiments, observations, data collection, and validation [112,113]. The selection of the appropriate research paradigm, and corresponding research methods (or their combination), depends on the research questions being answered and the availability of resources.

Analytical approaches make a very effective tool to understand phenomena, deduce results, and propose new algorithms in computer science. Particularly, formal methods are very useful with respect to problems where established mathematical tools, such as queuing theory [114–116], graph theory [117–119], or network calculus [120], can be directly applied to dissect and solve the issues in hand. As routing is the main subject of this thesis, graph theory is particularly interesting as it has been extensively used in the literature for formally defining topologies and routing algorithms [69, 72, 73, 75, 76, 78]. A major drawback of the analytical approaches, however, is that mathematical abstraction of large complex systems with dynamic behavior, such as HPC clouds, becomes very complex. Simplified abstraction techniques, even though keep mathematical models simple, are prone to ignoring critical issues affecting the implementation and operation of the system [121]. In this thesis, analytical approaches are used to address parts of our research questions, *RQ1*, *RQ2*, and *RQ5*. Specifically, we modeled the impact of node-oblivious routing and network sharing on the routing performance in large fat-trees through probabilistic methods. Such methods are useful in estimating the impact in a real-world system which can work in a variety of scenarios, and with various configurations. Similarly, the implications of routing transition on network reconfiguration cost and time are also modeled mathematically. In addition, formal graph theory is also used, where applicable, to formally define proposed routing approaches in the research papers produced as part of the work leading to this thesis.

In the context of networking research, three experimental research methods are commonly employed: *Empirical measurements*, *hardware experiments*, and *simulations* [111, 122, 123]. Empirical measurements are often used to understand the properties of an existing system. Both *exploratory* [124] and *explanatory* [122] empirical methods are common. When using the *exploratory* methods, observations like network measurements are performed without affecting the network being observed in any way. Such methods are useful to understand properties of a large real-world system, such as a supercomputer, without modifying its configurations. Access to such a large system was not available when the work in this thesis was carried out so exploratory research methods are not employed. *Explanatory* research, however, is concerned with quantifying relationship among different configurations and parameters through controlled experiments [122]. To come up with the research questions addressed in this thesis, we deduced the shortcomings of IB as an HPC cloud interconnect through small scale IB experiments with different configurations and observed the performance results. Then, to address the research questions, *hardware experiments* and *simulations* are the two major experimental research methods employed in this thesis, and thus, are detailed in separate subsections in the following.

1.3.1 Hardware Experiments

Hardware experiments are very useful for effectively validating the suitability of a proposed solution in the real-world systems. In *constructive research* [125], experiments are generally conducted through *prototype* software and hardware implementation. In many cases, the purpose of experiments is to compare the performance of the newly proposed solutions with the existing state-of-the-art systems [126]. In this thesis, for example, while addressing

RQ1, *RQ2*, and *RQ5*, we compared our modified fat-tree routing algorithm with the de facto fat-tree routing available on IB systems through hardware experiments. Even when such a comparison is not possible because no comparable system is available, hardware experiments are still valuable to confirm that the proposed solutions are practically applicable in real-world scenarios. This was the case with our proposed solutions to address *RQ3*, and *RQ6*. Specifically, the proposed inclusion of provider-defined policies in the routing decision process, and our self-adaptive HPC network architecture, was not directly comparable with any existing systems.

Understandably, hardware prototype implementations are expensive. Software prototypes, however, if requiring no specific change in the hardware, are less expensive to build. This doctoral work was conducted as a part of the ERAC project[9], where principles of *canonical action research* [127] were employed to collaborate effectively with the industrial partners. The project was driven by the goal of building small working prototypes that meet the requirements identified by each of the research questions. Most of the work presented in this thesis required no change in the hardware for the implementation, and was demonstrated on a local small-scale test-bed using readily available IB hardware and the state-of-the-art OFED software stack. Addressing *RQ4*, though, requires a specialized hardware architecture implementing the vSwitch model, which is not available. To cater for this issue, we used an *emulation* technique where vSwitches in the proposed architecture were emulated by means of real IB switches, for the purpose of hardware experiments.

Benchmarking is often used to test and compare results for two systems using a standard set of testing criteria. Several benchmarks are used throughout this doctoral work to evaluate our implementations, including OFED's IB performance testing utility (*perftest*), the *HPC Challenge Benchmark Suite* (HPCC) [128], the *OSU Micro benchmarks* [129], the *Netgauge* performance measurement toolkit [130], and the *NAS parallel benchmark suite* (NPB) [131] for evaluating HPC application performance.

A challenge of hardware experiments is that large-scale experiments, such as those with hundreds of end nodes and switches, are not possible unless access to such large amounts of hardware is actually available. However, large-scale experiments are important to validate scalability of proposed solutions. To address this challenge, simulations are used for the work in this thesis.

1.3.2 Simulations

In the simulation-based research method, an abstract model is designed and mirrored in a software *simulator*, capturing all relevant properties of the studied system. In this way, the simulations, when executed, represent behavior imitating the real system. Simulation is a very flexible and extensively used method to model and evaluate new systems and solutions. Simulations have been extensively used in the work related to this thesis to evaluate the performance of new proposed routing algorithms, reconfiguration methods, and software

[9]The ERAC (Efficient and Robust Architecture for the Big Data Cloud) was funded by the Norwegian Research Council under the project number 213283/O70. The main objective of the ERAC project was to provide the knowledge and solutions that enable an elastic, scalable, robust, flexible, secure, and energy-efficient cloud architecture for the future Internet.

solutions at large scale, and compare them with the existing solutions. That is, simulation is used as a research method when addressing all the research questions of this thesis.

An important consideration while designing simulations is the level of details needed for a particular evaluation. For instance, for evaluating the performance of a switching algorithm or routing algorithm against a very specific workload, *flit-level* simulations are useful. Flit-level simulations model data path information in the network at the transfer level of a flit, and hence, are accurate but, at the same time, computationally expensive to run. On the other hand, for generalized routing evaluation against common traffic patterns and randomized workloads, metrics deductible from the inspection of the routing tables, such as maximal congestion at each link, are sufficient. To address *RQ1*, we used flit-level simulations as an evaluation of our new load-balancing technique that requires different traffic roles for the nodes in the network. For other research questions, we used a light-weight routing congestion simulator.

Flit-Level Simulations

For the work in this thesis, we used an extended IB flit-level simulation model [132], originally contributed by *Mellanox Technologies* [133]. The model is implemented using the OMNeT++ network simulation framework [134, 135] and has been used extensively in the literature to evaluate IB networks [136–143]. To connect the flit-level simulations with the OFED software stack, we developed a custom *tool-chain* based on IB network topology simulator.

The InfiniBand Network Simulator (ibsim)

Throughout this work, we use OFED's IB network simulator, *ibsim*, to emulate physical topologies. The ibsim simulator emulates the fabric behavior by using MAD communication with the OpenSM, and is distributed with the OFED software stack.

The Oblivious Routing Congestion Simulator (ORCS)

The ORCS [144] is used to study the performance of the routing algorithms proposed in this thesis. The ORCS is capable of simulating a variety of communication patterns on statically routed networks [76, 145]. We extended ORCS to make it able to run patterns within partition boundaries, an improvement necessary to evaluate solutions related to the research questions *RQ2*, *RQ3*, and *RQ6*. The ORCS supports several metrics to reduce the data obtained as congestion maps in a single result. For example, the metric *sum_max_cong* represents the maximal congestion that occurred on any used route in each level. Another metric, *dep_max_delay*, is used to study the impact on one communication pattern, running in one group on nodes, due to another communication pattern that is being run in a second and different group of nodes. The simulator examines the congestion in only the first pattern, and reports the delay experienced by the victim pattern, caused by the interference from the communication in the other pattern. More details about the communication patterns and metrics supported by ORCS are given in [144] and [146].

1.4 Thesis Outline

This thesis is presented as a collection of research papers published in different journals and conferences during the doctoral studies. The thesis is structured into two parts: Part I '*Overview*' and Part II '*Research Papers*'. In Part I, research results achieved in this thesis, based on the published research papers, are collectively presented. The full text versions of all the research papers produced are annexed in Part II. There are four chapters in the first part. In the current chapter, Chapter 1, *Introduction*, we motivate our work, provide research questions, and enlist research methods employed to carry out the research presented in this thesis. Technical background needed to understand the concepts and solutions presented in the thesis is given in Chapter 2, *Background*. In Chapter 3, *Contributions and Summary of Research Papers*, the main contributions of the thesis are showcased, together with summaries of all the research papers produced as part of this doctoral work. Finally, Chapter 4, *Closing Remarks and Future Work*, concludes the thesis overview and point out some opportunities of the future work. Part II provides full-text versions of seven research papers.

1.5 Published Work

As part of this doctoral work in total four journal publications and three conference papers were produced. In addition, due to the industrial applicability of the doctoral work, many of the novel concepts and methods from the research papers are filed as patent applications. Last, but not least, the doctoral work was also presented as a *Doctoral Showcase* in the International Conference for High Performance Computing, Networking, Storage and Analysis (SC, also known as *Supercomputing*), 2016.

Conferences and Journal Publications

Paper I	**A Weighted Fat-Tree Routing Algorithm for Efficient Load-Balancing in InfiniBand Enterprise Clusters [147]**
Published at	23rd *Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), 2015.*
Authors	Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen and Tor Skeie

Paper II	**Partition-Aware Routing to Improve Network Isolation in InfiniBand Based Multi-tenant Clusters [148]**
Published at	15th *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2015.*
Authors	Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen and Tor Skeie

Paper III	**Efficient network isolation and load balancing in multi-tenant HPC clusters [149]**
Published in	*Future Generation Computer Systems (FGCS), Volume 72, July 2017.*
Authors	Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen and Tor Skeie

Paper IV	**SlimUpdate: Minimal Routing Update for Performance-Based Reconfigurations in Fat-Trees [150]**
Published at	1st *IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), held in conjunction with IEEE International Conference on Cluster Computing (CLUSTER), 2015.*
Authors	Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen and Tor Skeie

Paper V	**Compact network reconfiguration in fat-trees [151]**
Published in	*The Journal of Supercomputing (JSC), Volume 72, Issue 12, Springer, 2016.*
Authors	Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen, Tor Skeie and Evangelos Tasoulas

Paper VI	**Efficient Routing and Reconfiguration in Virtualized HPC Environments with vSwitch-enabled Lossless Networks [152]**
Submitted to	*Concurrency and Computation: Practice & Experience (CONCURRENCY), Wiley, 2017.*
Authors	Evangelos Tasoulas, Feroz Zahid, Ernst Gunnar Gran, Kyrre Begnum, Bjørn Dag Johnsen and Tor Skeie

Paper VII	**A Self-Adaptive Network for HPC Clouds: Architecture, Framework, and Implementation [153]**
Submitted to	*IEEE Transactions on Parallel and Distributed Systems (TPDS), 2017.*
Authors	Feroz Zahid, Amir Taherkordi, Ernst Gunnar Gran, Tor Skeie and Bjørn Dag Johnsen

Patents

- **US20160014049**, System and method for supporting efficient load-balancing in a high performance computing (hpc) environment.
  Inventors: *Feroz Zahid*, Ernst Gunnar Gran, Bartosz Bogdanski, Bjørn Dag Johnsen
  Filed on: *06.07.2015*, Published on: *14.01.2016*
- **US20160127236**, System and method for supporting partition-aware routing in a multi-tenant cluster environment.
  Inventors: *Feroz Zahid*, Ernst Gunnar Gran, Bartosz Bogdanski, Bjørn Dag Johnsen
  Filed on: *29.10.2015*, Published on: *05.05.2016*
- **US20160277232**, System and method for efficient network reconfiguration in fat-trees.
  Inventors: Bartosz Bogdanski, Bjørn Dag Johnsen, *Feroz Zahid*, Ernst Gunnar Gran
  Filed on: *17.03.2016*, Published on: *22.09.2016*
- **US20160301565**, System and method for efficient network reconfiguration in fat-trees.
  Inventors: *Feroz Zahid*, Bartosz Bogdanski, Bjørn Dag Johnsen, Ernst Gunnar Gran
  Filed on: *23.01.2016*, Published on: *13.10.2016*
- **US20170104682**, System and method for efficient network isolation and load balancing in a multi-tenant cluster environment.
  Inventors: *Feroz Zahid*, Ernst Gunnar Gran, Bartosz Bogdanski, Bjørn Dag Johnsen
  Filed on: *14.06.2016*, Published on: *13.04.2017*
- **US20170104817**, System and method for efficient network isolation and load balancing in a multi-tenant cluster environment.
  Inventors: *Feroz Zahid*, Ernst Gunnar Gran, Bartosz Bogdanski, Bjørn Dag Johnsen
  Filed on: *14.06.2016*, Published on: *13.04.2017*
- **US20170149887**, System and method for efficient virtualization in lossless interconnection networks.
  Inventors: Evangelos Tasoulas, *Feroz Zahid*, Bjørn Dag Johnsen, Ernst Gunnar Gran
  Filed on: *14.07.2016*, Published on: *25.05.2017*
- **US20170149888**, System and method for efficient virtualization in lossless interconnection networks.
  Inventors: Evangelos Tasoulas, *Feroz Zahid*, Bjørn Dag Johnsen, Ernst Gunnar Gran
  Filed on: *14.07.2016*, Published on: *25.05.2017*

Other Work

Doctoral Showcase	**Realizing a Self-Adaptive Network Architecture for HPC Clouds**
Published at	*The International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing, SC), 2016.*
Authors	Feroz Zahid, Ernst Gunnar Gran, Tor Skeie

Chapter 2

Background

This chapter provides the necessary technical background to understand the rest of this thesis. The chapter starts with an introduction to high-performance computing with a focus on interconnection networks. Next, an overview of cloud computing is given. After that, we provide an introduction to the IB architecture, which is the interconnection technology we use for demonstrating concepts and solutions presented in this thesis. Last, a brief overview self-adaptive systems is presented towards the end of the chapter.

2.1 High-Performance Computing

Generally speaking, the term high-performance computing (HPC) or *supercomputing* refers to any computational implementation involving aggregating computer power to get higher performance than what is offered by general-purpose computers. The first use of the term *'supercomputer'* goes back to the 1920*s* for referring to the IBM tabulators at Columbia University [154]. However, *CDC 6600* [155], released in 1964, is largely agreed to be the first real supercomputer. Today, the computational power provided by HPC is fueling advances in areas as diverse as astronomy, cosmology, engineering, life sciences, and business intelligence [2–5]. More recently, the exploitation of HPC technology in cutting-edge applications of machine learning, AI, and big data analytics, has reiterated the importance of HPC as an instrumental tool for the continued development in science and technology [64].

While CDC 6600 had only a single CPU running at just 10MHz with less than one MB of main memory at its disposal, today's fastest supercomputers are powered by thousands of interconnected nodes, each with hundreds of processing cores, possessing staggering computing performance. For instance, Sunway TaihuLight [83], the fastest supercomputer as of the writing of this thesis, has more than 10 million processing cores. The TaihuLight is capable of performing about 93014.6 teraFLOPS[1]. Regardless of the architecture, processing power from distributed nodes in such large machines can only be extracted by the applications when complemented by a powerful interconnection network connecting the available processors. The topic of this thesis is rooted in employing HPC interconnect technologies in a cloud to help extending HPC power to a broader audience and applications.

[1]Referring to TaihuLights's *Maximal achieved performance*, R_{max}, as measured by the LINPACK Benchmark [156]. The LINPACK benchmark is used to rank supercomputers in the Top500 list [1].

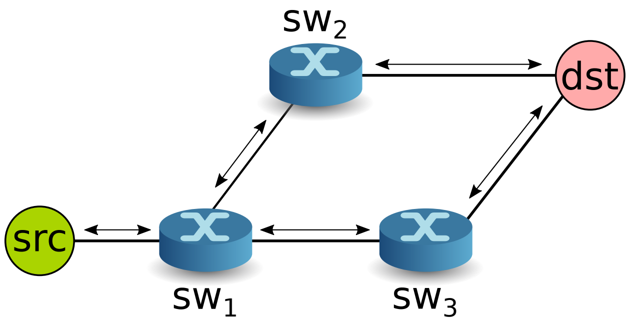


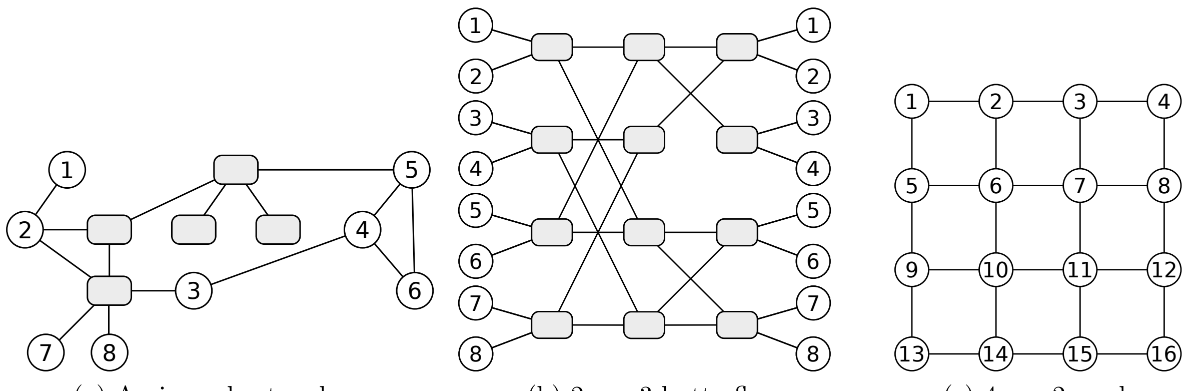
Figure 2.1: An example interconnection network arranged in a topology with two processing nodes, three switching elements, and five communication channels. The path network packets take, from the source node *src* to the destination node *dst*, is determined by the routing algorithm.

2.2 Interconnection Networks

Communication is an essential building block of any digital system. An interconnection network enables communication, and can be defined as a *programmable system that transports data between end points in a digital system* [88]. Interconnection networks are used in a variety of digital systems both for connecting elements within a computing system, such as processors and memory components, and for connecting external computers together, like in a local area network (LAN). In this thesis, the focus is on the interconnection networks employed in HPC systems and supercomputers to connect external processing and storage systems together. Interconnection networks play a critical role in determining the overall performance of any HPC system and the applications running on it [69,88]. Modern supercomputers typically rely on *lossless* interconnection networks to minimize communication overhead, a necessity to achieve required high-performance communication between the sub-systems. The lossless interconnection networks, contrary to the lossy networks, do not drop packets in normal network operations by employing a flow-control mechanism. The flow control mechanism ensures that the network packets are only forwarded between a *source* and a *destination* when the source is ensured that the destination has adequate resources available to handle the packet communication. IB, which is the demonstration interconnection network of choice in this thesis, defines a lossless interconnection network (IB is further detailed in Section 2.4).

An interconnection network can be defined as a directed multigraph with vertices of the graph representing the *network nodes* and the edges representing *communication channels*. The communication channels represent the transmission medium, such as a wire, through which data communication, typically in the form of network *packets*, move from one end to the other. The network nodes are of two types: *end nodes* or compute nodes which generate, consume, process, and store data, and the *switching elements*, such as routers and switches, whose main role is to forward data in the network on behalf of the end nodes.

The performance of an interconnection network is largely characterized by its *topology*, *routing algorithm*, and *switching* layer techniques. The network topology defines the static physical arrangement in which the network nodes and channels are interconnected. There may be more than one data *path*, defined by the sequence of intermediate channels, between a particular source and a destination node in the topology. Consider an example interconnection network, as shown in Figure 2.1. The packets from the source node, *src*, to



(a) An irregular topology (b) 2-ary-3-butterfly (c) 4-ary-2-mesh

Figure 2.2: A regular topology follows a specific graph structure (*b* and *c*) while irregular topologies (*a*) do not.

the destination node, *dst*, can take either the path $src \rightarrow sw_1 \rightarrow sw_2 \rightarrow dst$ or the path $src \rightarrow sw_1 \rightarrow sw_3 \rightarrow dst$. The job of the routing algorithm is to calculate candidate output channels at intermediate switching elements, and thus, determine the path a packet takes through the network. The switching layer is characterized by the techniques implemented in the switches, such as how buffers in the switches are allocated and released, and how and when ingress and egress ports of the switches are connected to forward packets [69]. Flow control mechanisms are also tightly coupled with the switching technique implemented by the switching layer. In the following, we discuss network topologies and routing in more detail, the two determinant characteristics of the interconnection networks relevant to this thesis.

2.2.1 Network Topologies

As mentioned above, a network topology determines the arrangement of the nodes and channels in a network. A *regular* topology is characterized by a specific graph structure, such as a *ring*, *tree*, or *mesh*, while irregular topologies have no such structure[2], as shown in Figure 2.2. The topology makes an important factor affecting both the cost and the performance of an interconnection network. In addition, other determinant performance factors, such as routing, also depend on the network topology. The setup cost of a network is largely determined by the number of switching elements, required switch *radix* (number of ports on a switch), switch complexity, and the cabling needed for the specific arrangement a topology specifies. The performance, on the other hand, is determined by the combination of the topology, routing algorithm, switching layer techniques, and the application workload [69]. Two important performance criteria are *latency* and *bandwidth*. The former is defined by the time required by a packet to reach from a particular source to a destination in the network; while the later describes the amount of data transferable between two end-points in a unit of time. As both the latency and the bandwidth depend on other factors such as routing, switching, and applications, other topology-specific metrics like network diameter and channel load are used to reflect on the performance of a topology regardless of other

[2]Regular topologies, however, can become *semi-irregular* or hard to discover due to faults in the network [140].

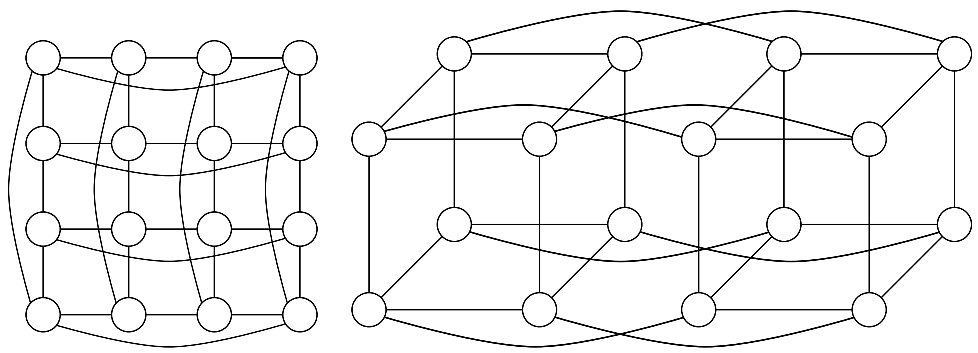

(a) 4-ary-2-cube (b) 2-ary-4-cube

Figure 2.3: Example *k-ary-n-cube* topologies

factors [157].

Following the classification of interconnection networks in [158] and [69], we categorize topologies into four main categories: *shared-medium*, *direct*, *indirect*, and *hybrid* topologies, presented in the following.

2.2.1.1 Shared-Medium Systems

In shared-medium networks, each communicating node in the topology is connected to a shared transmission medium, also called *bus*. There are two basic ways a medium can be shared between communicating nodes, *time sharing* and *frequency sharing*. In a time sharing shared-medium system, only one node is allowed to use the medium at a time. Frequency sharing divides up the available frequency range among the transmitting nodes in the shared medium allowing concurrent connections [159]. In both cases, however, the basic problem of arbitration of the use of the shared medium remains the same. Many shared-medium systems, such as the original Ethernet, use *broadcasting*. In broadcast communication, each node receives the traffic sent to the shared medium, however, only the destination node interprets it. In the absence of strict mastership arbitration between nodes, which is practically inefficient, large amount of packet collisions may occur, resulting in high packet loss. Shared-medium networks' limited scalability and efficiency restrict their use in HPC systems. In this thesis, we will not consider shared-medium networks further.

2.2.1.2 Direct Topologies

In direct topologies, all the network nodes act both as end nodes and switches, and each node is connected directly to a subset of other nodes in the network. In this way, two nodes can communicate by transferring packets via intermediary nodes even if there is no direct connection between them. Direct topologies are popular in HPC systems, and a variety of such topologies exist. The most popular direct topologies follow a *k-ary-n-cube* definition [160], with n dimensions and k nodes in each dimension connected in a ring. The class of *k-ary-n-cubes* and its variants form many important HPC topologies, such as *torus*, *hybercube*, and *mesh*[3]. For example, *Titan* [161], and the *K Computer* [162] on Top500

[3]The mesh is a variant of *k-ary-n-cube* in which k nodes in each of the dimensions are connected without wraparound links.

list [1] use a torus topology.

The *k-ary-n-cubes* can be constructed recursively by adding k *k-ary-(n-1)-cubes* in a way that corresponding nodes from the different dimensions are interconnected in a new ring to form the k dimension version. For each position $i \in \{0, ..., k^{n-1} - 1\}$, edges between composite *k-ary-n-cubes* are created by connecting all the k ith position nodes in the ring [163]. Figure 2.3 shows two examples of *k-ary-n-cube* topologies.

The direct topologies provide a high degree of path diversity to enable fault-tolerance. Moreover, such topologies exhibit excellent performance for HPC applications where communication is mainly between neighboring nodes. A disadvantage with the direct topologies is that, as the number of dimensions is added, both the length of the links and required switch radix increase, making them less attractive in very large systems.

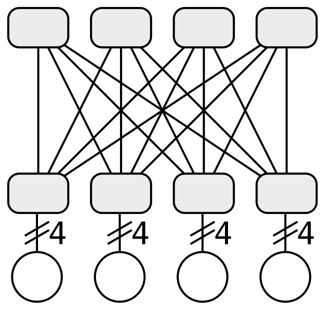
2.2.1.3 Indirect Topologies

Indirect topologies are the topologies used to demonstrate the solutions in this thesis. Indirect topologies are characterized by dedicated switching nodes and end nodes in the network. A popular indirect topology in HPC systems is the family of fat-tree topology [71], which is a subset of indirect topologies often referred to as multistage interconnection networks (MINs) [164]. In MINs, connections between the compute nodes are formed through a number of switch stages. Apart from fat-trees, *Cross-Bars*, Clos Networks [165], and *butterfly* networks are other popular MINs. We now discuss fat-tree topology in detail, which is the network topology used in this work.

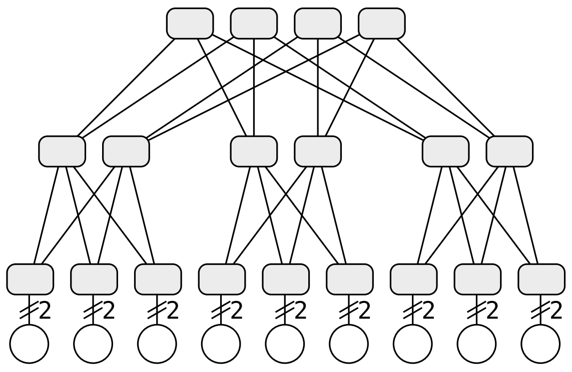
Fat-Trees

The fat-tree topology was first introduced by Leiserson [71] in 1985 , and is a class of general-purpose network topologies characterized by a multi-stage hierarchical tree-like structure, commonly with multiple *root* nodes and end nodes at the leaves of the tree. The initial idea behind fat-trees was to employ *fatter* links, that is links with more available bandwidth between nodes, as we move close to the roots of the topology. The fatter links help to avoid congestion in the upper-level switches and the *cross bisection-bandwidth* (CBB) is maintained. The CBB is defined as the worst-case bandwidth available between the segmented partitions when the network is cut into two halves [166]. Different variations of fat-trees have been presented in the literature through the years, including k-ary-n-trees [167], Generalized Fat-Trees (GFTs), Extended Generalized Fat-Trees (XGFTs) [168], Reduced Unidirectinal Fat-Trees (RUFTs) [169], Parallel Ports Generalized Fat-Trees (PGFTs), Real Life Fat-Trees (RLFTs) [87, 141], and Quasi Fat-Trees [170], among others. Some example fat-trees are shown in Figure 2.4.

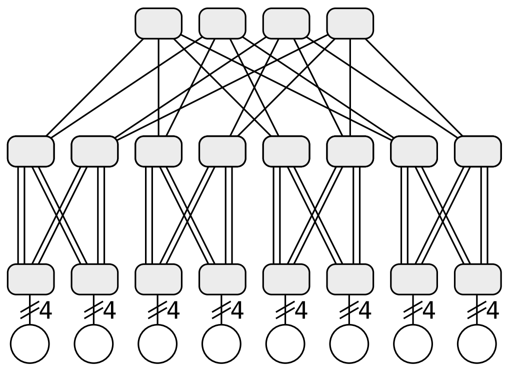
Many supercomputers in Top500 list [1] employ a fat-tree topology to take advantage of the useful properties fat-trees offer [82, 171, 172]. These properties include full bisection-bandwidth and inherent fault-tolerance due to the existence of multiple routes between any two end nodes. Moreover, fat-trees are easy to build using commodity switches [173]. Fat-Trees can be constructed using several different notations. A k-ary-n-tree [167] is an n level fat-tree with k^n end nodes and $n \times k^{n-1}$ switches, each with $2k$ ports. Each switch has an equal number of up and down connections in the tree, except for the root switches, which do not have upward connection. The k-ary-n-tree fat-tree notation is very restricted and



(a) 4-ary-2-tree



(b) XGFT(3;3,3,2;1,2,2)



(c) PGFT(3;4,2,4;1,2,2;1,2,1)

Figure 2.4: Example fat-tree topologies

is not able to define a lot of real-world fat-trees. Moreover half of the ports of the top-level switches are not used. A variant, the m-port-n-tree, can be created by utilizing the remaining k ports of the top-level switches to connect to a copy of $(n-1)$-level sub-fat-tree. In this way, the m-port-n-tree definition represents the fully connected k-ary-n-tree. The GFT notation, $GFT(h, m, w)$, releases the restrictions of having the same number of upgoing and downgoing links at each intermediate level switch. A $GFT(h, m, w)$ has $h + 1$ levels of nodes with m downward connections and w upward connections at all levels, except for the end nodes that do not have children and the root switches that do not have parent nodes. The XGFT definition further generalizes the GFT notation and allows both different number of up and down connections for the switches, and different number of connections at each level in the tree. An $XGFT(h; m_1, ..., m_h; w_1, ..., w_h)$ is a fat-tree with $h + 1$ level of nodes. Levels are denoted from 0 to h, with end nodes at the lowest level 0, and switches at all other levels. Except for the end nodes that do not have children, all nodes at level i, $1 \leq i \leq h$, have m_i child nodes. Similarly, except for the root switches that do not have parents, all other nodes at level i, $0 \leq i \leq h-1$, have w_{i+1} parent nodes. A problem with the XGFT notation is that multiple links between switches cannot be represented. The PGFT notation, $PGFT(h; m_1, ..., m_h; w_1, ..., w_h; p_1, ..., p_h)$ permits multiple connections between switches. A large variety of topologies can be defined using XGFTs and PGFTs. However, for practical purposes, RLFT, which is a restricted version of PGFT, is introduced to define fat-trees commonly found in today's HPC clusters [141]. A RLFT uses the same port-count switches at all levels in the fat-tree. A summary of the basic characteristics, number of end nodes and number of switching elements, of some of the most popular notations is provided in Table 2.1.

Table 2.1: Fat-Tree Notations

Notation	#End Nodes	#Switches
k-ary-n-tree	k^n	$n(k^{n-1})$
m-port-n-tree	$2(\frac{m}{2})^n$	$(2n-1)(\frac{m}{2})^{n-1}$
$GFT(h, m, w)$	m^h	$\sum_{i=1}^{h} m^{h-1} w^i$
$XGFT(h; m_1, ..., m_h; w_1, ..., w_h)$	$\prod_{i=1}^{h} m_i$	$\sum_{i=1}^{h}(\prod_{j=i+1}^{h} m_j \prod_{j=1}^{i} w_j)$

2.2.1.4 Hybrid Topologies

A hybrid network topology is formed by any combination of shared-medium, direct, and indirect topologies [158]. The basic motivation of such a combination generally is that they can reduce the node distance as compared to the direct or indirect network topologies, while providing higher bandwidth and efficiency than the shared-medium networks [69, p. 34]. Some examples of hybrid topologies include *Long Hop* networks [174], *dragonfly* [175], and *SlimFly* [157]. For instance, in dragonfly networks, the global topology is formed by combining smaller groups of direct or indirect topologies together through a global intra-group topology. Each group has at least one link directly connected to each other group.

2.2.2 Routing

A routing algorithm defines the path that a packet takes to move from a source node to a destination node in the network. While the characteristics of a topology define the best-case performance that can be achieved in a distributed system, it is actually the routing algorithm, together with the employed switching techniques, that determines the actual realization of this potential for a particular communication pattern [88, p. 159].

A large number of routing algorithms have been proposed in the literature, which can be classified according to different criteria. For example, a taxonomy of the routing algorithms is provided in [69, p. 140]. The most fundamental classification often used is based on the method a routing algorithm employs to select a particular path, from a source to a destination node, within a set of available alternative paths in the topology. A *deterministic* routing algorithm always selects the same path between a given pair of source and destination node. An *oblivious* routing algorithm, on the contrary, may supply one of several different available paths each time a packet is routed. However, the routing is done regardless of, or *oblivious* to, the current network conditions. *Adaptive* routing alters or *adapts* the routing decisions according to the network state, and hence, tends to avoid congested links in the network. From this basic information, it may seem as if adaptive routing should be the method of choice in interconnection networks. However, it is the deterministic routing which is most commonly found in HPC systems because of its simplicity offering both speed and inexpensive switch implementation as well as in-order delivery of packets (which is crucial for some applications). In this thesis, we are mainly confined to deterministic routing algorithms. The main reason to choose deterministic routing over adaptive routing is that, in IB networks, adaptive routing is not readily available. Even though there are some academic and proprietary implementations of adaptive routing [52, 94, 176], all of the nine available routing engines in OpenSM [76–78, 85, 86], as of this writing, use a deterministic routing approach. Another important reason to choose deterministic routing is the flexibility required for our implementations. Adaptive routing generally works solely on the basis of the network conditions, and adding complex routing criteria, for instance based on tenant SLAs, makes it very inefficient. Routing based on such criteria is better determined over a period of time through monitoring and evaluation and, once complemented with fast network reconfiguration mechanisms, deterministic routing is able to fulfill the requirements.

Topology-specific routing algorithms are designed for a particular topology. Such algorithms, in general, take the characteristics of the topology into account to efficiently calculate optimal or near-optimal paths in the network. The fat-tree routing algorithm [86, 87] is a topology-specific routing algorithm for fat-trees. *Topology-agnostic* routing algorithms, on the other hand, are designed to work on any topology. DFSSSP [76] and LASH [78] are two examples of topology-agnostic algorithms.

The OpenSM supports nine routing algorithms, including Up*/Down* [77], MinHop, DFSSSP [76], LASH [78], DOR [85], and the fat-tree routing algorithm [86, 87]. We now discuss the fat-tree routing algorithm further.

Fat-Tree Routing Algorithm

The fat-tree routing algorithm [86, 87] is one of the most popular routing algorithms for the fat-tree topologies in IB networks. The algorithm supports the routing of various types of fat-trees, including *k-ary-n-trees*, also with non-constant *k*, that allows for different *arity* switches at different levels in the fat-tree. In addition, the fat-tree routing algorithm also works for fat-trees that are not fully populated, a feature useful when routing degraded fat-trees due to network faults. Later work extends the fat-tree routing algorithm for more generalized RLFTs [87]. Furthermore, several improvements to the fat-tree routing, such as making it work for imbalanced and degraded fat-trees and supporting intra-subnet routing, were proposed in [177]. The fat-tree routing provides non-blocking traffic for *shift permutation* traffic patterns. These permutation patterns are performance benchmarks as they are often employed in *all-to-all* message communication common for many HPC applications.

The fat-tree routing algorithm aims to generate routing tables that evenly spread shortest-path routes across the links in the fat-tree topology. The algorithm traverses the fabric in the *indexing order* and assigns target routes to the end nodes at each switch port. For the end nodes connected to the same leaf switch, the indexing order depends on the switch port to which the end node is connected (port numbering sequence). For each switch port, the algorithm maintains a *port usage counter* and uses it to select the least-used port each time a route is added (if more than one option is available). If there are multiple ports connecting the same two adjacent switches, the ports form a *port group*. In such a case, the least-used port of all the available port groups is selected to add a new route. The port assignment to the end nodes, at the switches throughout the network, is performed recursively in two stages, starting at the leaf switches. In the first stage, the algorithm traverses from each end node up towards a tree root, allocating the downgoing port and the corresponding route to the end node. After the downgoing ports are set, the algorithm assigns upward ports to the end nodes on all the connected downward switches by descending down the tree. The process is then repeated recursively by moving up to the next level of the tree. The pseudo-code of the algorithm is provided in Listing 1.

Listing 1 The Fat-tree Routing Algorithm

```
Require: Topology has been discovered
Ensure: Well-balanced routing tables are generated
for each switch in leaf switches do
    for each node connected to switch in indexing order do
        set route to local switch port for the node
        RouteDowngoingByAscending(switch,node)
    end for
end for
procedure RouteDowngoingByAscending(switch, node)
    find the least loaded port of all the upgoing port groups
    assign route to node at the connected remote_switch to that port
    increase port counter
    RouteUpgoingByDescending(switch,node)
    RouteDowngoingByAscending(remote_switch,node)
end procedure
procedure RouteUpgoingByDescending(switch, node)
    for each groups in downgoing groups do
        select remote_port with the minimum counter
        assign route to node at the remote_port on remote_switch
        increase port counter
        RouteUpgoingByDescending(remote_switch, node)
    end for
end procedure
```

2.3 Cloud Computing

Over the last several years, *Cloud Computing* has brought a paradigm shift in computing. An increasing number of enterprises, organizations, and individuals are relying on services offered by cloud systems to meet their computational and storage demands. The trend can be seen by *Gartner's* cloud computing forecasts of 2017 [178]. Gartner predicts that the public cloud services market is set to grow up to 380 billion dollars by 2020. In addition, they forecast that the cloud adoption strategies will influence more than 50 percent of IT outsourcing deals in 2017.

According to the definition proposed by the National Institute of Standards and Technology (US):

> '*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*' [13]

However, as with all other *buzzwords*, cloud computing can be better defined through its general properties. We start from the *on-demand availability*, which specifies that the resources offered by a cloud computing platform are available *on demand*. That is, the resources are not acquired by the cloud users until they require them. Next, an important aspect of cloud computing systems is that the services to the clients are offered *over the* *network*. Physical resources are owned and operated by the cloud owners and a network access is provided to the cloud users, often with higher-level of control and usage freedom. The cloud computing model provides an illusion of *infinite* resource availability to its users. In fact, a large *shared pool of resources* is available. From this pool, users provision the resources they require elastically. *Elastic provisioning* is the feature of cloud computing by which users can acquire more resources whenever and for what period of time they require them. Resources that are no longer required by a user are released back to the shared pool. Finally, although not necessarily a basic property of a cloud service, the most important feature that has contributed drastically to the popularity of cloud computing systems is its generally used business model. Tenants only pay for the time period for which they acquire and use resources i.e. *pay-as-you-go* [179]. This is drastically different from the classic business model of the cluster computers where resources are set aside by the users to sustain the performance in peak-load scenarios. The pay-as-you-go model significantly reduces capital expenditures for the cloud computing users.

Broadly speaking, clouds come in three flavors: *public*, *private*, and *hybrid* clouds. The public clouds, as the name suggests, offer infrastructure and services to their customers over the Internet. The basic advantage of using a public cloud, as mentioned above, is that the organizations need not to invest a large capital expenditure to setup the hardware needed to run their applications and services. The private clouds, on the other hand, are owned and operated by a single organization, which can be thought as acting as both the cloud provider and the cloud user. Private clouds are used to efficiently utilize available resources shared among different applications and services owned by the same organization or a small group of organizations. A hybrid cloud is a combination of private and public clouds in which resources acquired from public clouds are used to complement the available hardware in the private infrastructure. For example, hybrid cloud setups can dynamically utilize public clouds for application cloud *bursting* [180] in high-load situations.

There are three basic cloud computing service models in use today:

- *Software as a Service (SaaS):* SaaS is the cloud computing model to provide applications over the Internet on-demand. There are many established and startup companies offering software as a web service to their customers and the list is building up every day. Recently, Microsoft offered the stable release of its flagship office products over the Internet as Office365[4]. Other examples include Salesforce[5] and GoogleApps[6].

- *Platform as a Service (PaaS):* PaaS is the model of provisioning system layer resources to the users, including operating system support and software development frameworks. Popular examples include Google AppEngine, AWS Elastic Beanstalk, and Heroku.

- *Infrastructure as a Service (IaaS):* IaaS refers to the cloud computing model for on-demand provisioning of infrastructural resources, typically in terms of VMs or containers. Popular IaaS service include Amazon EC2, Rackspace, and Microsoft Azure.

[4]http://office.microsoft.com/
[5]http://salesforce.com/
[6]http://apps.google.com/

Virtualization is a key technology behind cloud computing – it allows for the dynamic provisioning of applications, in terms of VMs or containers, over a large pool of physical resources in a cloud data center. Virtualization offers several practical benefits for a diverse set of applications, such as isolation, fast provisioning, scalability, and fault-tolerance [181, 182].

2.3.1 Are Clouds Ready for HPC?

Clouds offer off-the-shelf, effective, flexible and economical solutions that can be applied to almost any computational domain. For the computing needs of web services and business applications, cloud computing provides compelling benefits and help reducing up-front and operational costs remarkably. However, several challenges related to performance, availability, and security are hindering the pace of cloud computing progress in many other fields, for instance, in the area of HPC. In many cases, the network becomes the bottleneck that limits clouds from supporting applications that require higher degree of QoS. Most of the current public cloud solutions employ a network architecture, routing protocols, and security policies that are not designed to run HPC applications and other performance-demanding workloads. When tested under high-performance conditions, traditional network architectures are vulnerable to performance issues and violations of service level guarantees, typically needed for HPC and HPDA workloads. Studies conducted for the evaluation of HPC applications on public cloud computing systems have shown that cloud computing, at large, is yet not suitable for HPC [40–42, 49, 50].

InfiniBand, being a popular interconnect technology for HPC systems, such as supercomputers, promise a high potential of increasing the uptake of cloud computing for HPC applications. This thesis targets the realization of an efficient HPC cloud through the use of IB interconnection technology, in lieu of the traditional *best-effort* networking systems.

2.4 InfiniBand Architecture

The IB Architecture (IBA) [20] defines a serial point-to-point full-duplex interconnect supporting high-throughput and low latency I/O and interprocess communication. Thanks to the low CPU-overhead remote direct memory access (RDMA) supports, IB continues to be a very popular interconnect standard in supercomputing [1]. While the latest hardware available in the market today supports the *enhanced data rate* (EDR) of 100 Gb/s with 4x link widths, the future roadmap outlined by the association targets bandwidths reaching up to 200 Gb/s (*high data rate* - HDR) in 2017.

As shown in Figure 2.5, an IBA network consists of one or more *subnets* interconnected using routers. Within a subnet, hosts (processor end nodes and I/O devices) are connected using switches and point-to-point links. The network interface card (NIC) in IBA is called Host Channel Adapter (HCA). An HCA defines the point at which an IBA processor end node connects to the IBA network. A specialized version of the channel adapter, Target Channel Adapter (TCA), is used to connect I/O devices to an IBA fabric. The TCA includes an I/O controller specific to the I/O device's protocol, such as Fiber Channel (FC), Ethernet, or SCSI. Each channel adapter may have multiple ports with their own transmit

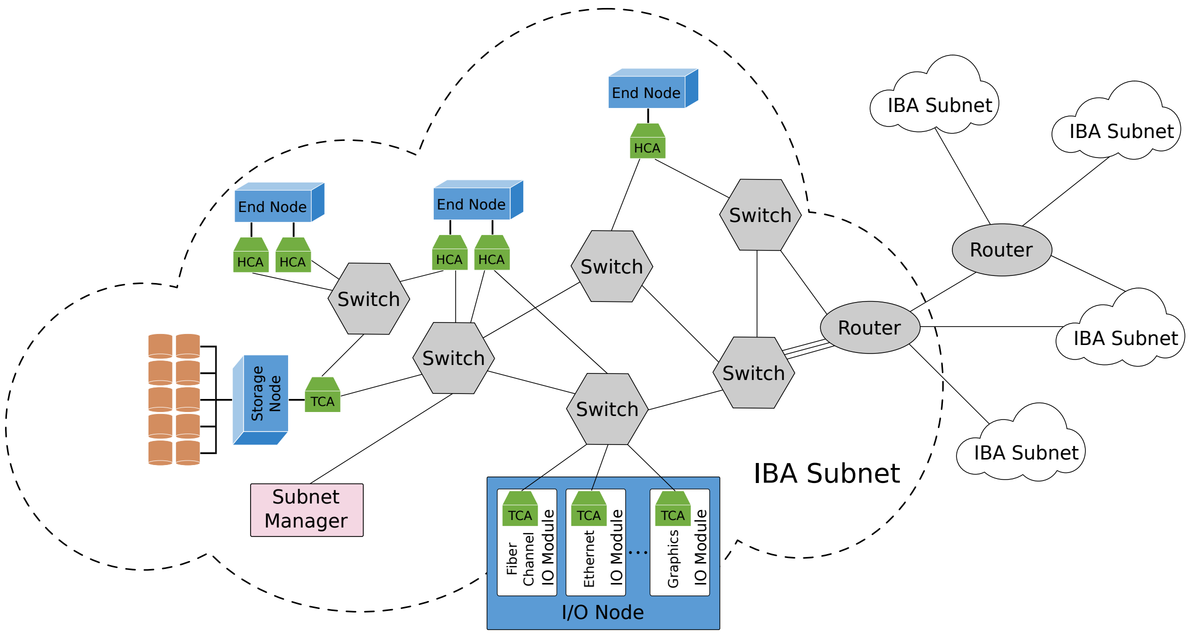


Figure 2.5: An IBA subnet consists of end nodes and I/O units connected through switches. Routers connects multiple subnets.

and receive buffers to enable concurrent communication. In this thesis, we will be mainly concerned with processor end nodes or *compute nodes* with HCAs.

There is one *master* management entity, as depicted in Figure 2.5, the *subnet manager* (SM)[7] - residing on any designated subnet device - that configures, and maintains the IBA subnet. Through a well-defined *subnet management interface* (SMI), the SM exchanges control packets, called *subnet management packets* (SMPs), with the *subnet management agent* (SMA) that reside on each IBA device. Using the SMI, the SM is able to discover the subnet topology, configure each channel adapter port, assign addresses, and receive failure notifications from the SMAs. The SM also performs periodic light sweeps of the subnet to detect any topology changes, like node addition/deletion or link failures, and reconfigures the network accordingly. More details about the subnet discovery mechanism in IBA are given in [183].

2.4.1 Addressing

In a subnet, each channel adapter port on the end nodes and all the switches are addressed using a unique 16-bit local identifier (LID) assigned by the SM. Switches use linear forwarding tables (LFTs) based on LIDs to route packets within a subnet. In addition, each IBA device has a globally unique identifier (GUID), called a node GUID, assigned by the hardware vendor. Similarly, each of the channel adapters and switch ports has a port GUID assigned by the vendor. Each port in the IBA subnet also has a 128-bit global identifier (GID) in the format of an IPv6 address, which is formed by combining a globally unique subnet prefix and the GUID of the port. Routers use GIDs to forward packets between

[7]There can be several SMs in the subnet for providing fault-tolerance. Except for the master SM, all other SMs are in standby mode. In case a master SM fails, a new master is negotiated by the standby SMs using a *master election and handover protocol* [20].

subnets. In a local subnet, the SM provides a GUID to LID/GID resolution service for node identification.

2.4.2 Routing

As mentioned above, intra-subnet routing in an IBA network is based on LFTs stored in the switches. Both unicast and multicast forwarding tables are supported[8]. However, the focus of this thesis is on routing based on unicast LFTs. Each entry in an LFT consists of a destination LID (DLID) and an output port. When a packet arrives at a switch, its output port is determined by looking up the designated DLID in the LFT of the switch. The LFTs are calculated by the SM using a routing algorithm, and installed in the switches as part of the subnet configuration process.

Packets destined to inter-subnet nodes have a global route header (GRH) specifying source and destination GIDs. Routers use the GRH to route packets between subnets. When the packet reaches the destination subnet end router, The LID associated with the Destination GID is used to forward the packet to the destination node in the subnet.

2.4.3 Communication

The IBA communication is asynchronous in nature powered by the notion of *Queueing*. The channel adapter provides communication interfaces, generally in the form of queue pairs (QPs), where *consumers* can queue up their service requests for send and receive operations[9]. The *send work queue* holds work requests dealing with the data that needs to be transferred to a remote node, while the *receive work queue* handles data that is received from a remote node[10]. The QPs are identified in a channel adapter using unique QP number. QP0 and QP1 are special QPs dealing with management traffic.

IBA defines both channel-based (Send/Receive) and memory (RDMA) semantics for the communication, and supports connection-oriented and datagram services.

2.4.4 Partitioning

Partitioning is a security mechanism provided by IBA to enforce isolation of logical groups of systems sharing a network fabric. The IB partitions provide similar isolation features as provided by Ethernet 802.1Q VLANs [184]. Each HCA port on an end node in the fabric can be a member of one or more partitions. Partition memberships are managed by a centralized partition manager, which is part of the SM. The SM configures partition membership information on each port as a table of 16-bit partition keys (*P_Keys*), where each partition is represented by a unique *P_Key*. The SM also configures switches and routers with the partition enforcement tables containing *P_Key* information associated with the LIDs.

[8]Unicast defines communication between a single sender and a single receiver. In contrast, communication between a single sender and multiple receivers is called multicast.

[9]The IBA supports up to 2^{24} QPs per channel adapter.

[10]Strictly speaking, the *remote* node can also be another *consumer* on the same node, such as a second channel adapter.

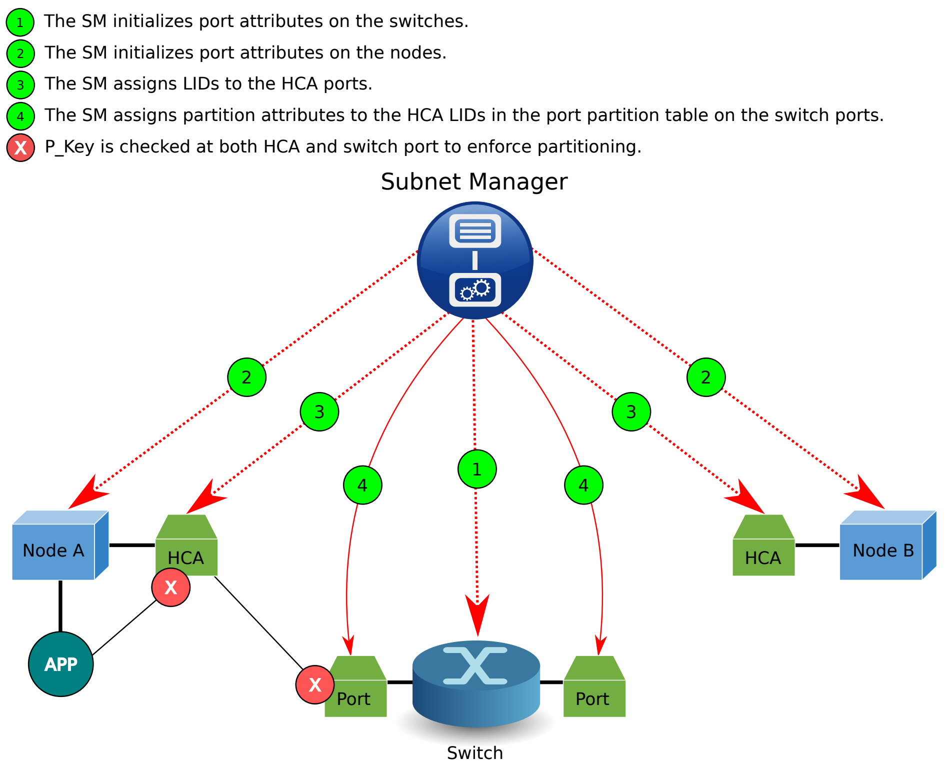

1 The SM initializes port attributes on the switches.
2 The SM initializes port attributes on the nodes.
3 The SM assigns LIDs to the HCA ports.
4 The SM assigns partition attributes to the HCA LIDs in the port partition table on the switch ports.
X P_Key is checked at both HCA and switch port to enforce partitioning.
Subnet Manager
2
3
4
1
4
3
2
Node A
HCA
X
APP
X
Port
Port
Switch
HCA
Node B

Figure 2.6: The IBA partition validation and enforcement process

For the communication between nodes, except for the management QPs, all other QPs are assigned to a particular partition. The *P_Key* information is then added to every IB transport packet sent. When a packet arrives at an HCA port or a switch, its *P_Key* value is validated against the table configured by the SM. If an invalid *P_Key* value is found, the packet is discarded immediately. Packet filtering is done at both HCAs and switches to ensure that a compromised node cannot violate the security of the system. In this way, communication is allowed only between ports sharing a partition. The IBA partition enforcement and validation process is shown in Figure 2.6.

Two types of partition memberships are supported: *full* and *limited*. Limited members cannot communicate with other limited members. However, limited members can communicate with the full members of the partition. Full members can communicate with all the members of a partition regardless of their membership type. There is a *default* partition that is created by SM regardless of the presence of other partitions and it only allows full membership.

2.4.5 Quality of Service

IBA is a layered architecture in which each physical link can be divided into multiple virtual links called virtual lanes (VLs). As shown in Figure 2.7, each VL has its own send and receive queues, together with separate flow-control and congestion management resources. In this ways, traffic in one VL does not block traffic in another VL on the same physical link. Quality-of-Service (QoS) is provided through a set of differentiated traffic classes, the Service Levels (SLs). The SL represents the class of service a packet will receive in the network, and the SL to which a packet belongs to is specified in the packet header. Each

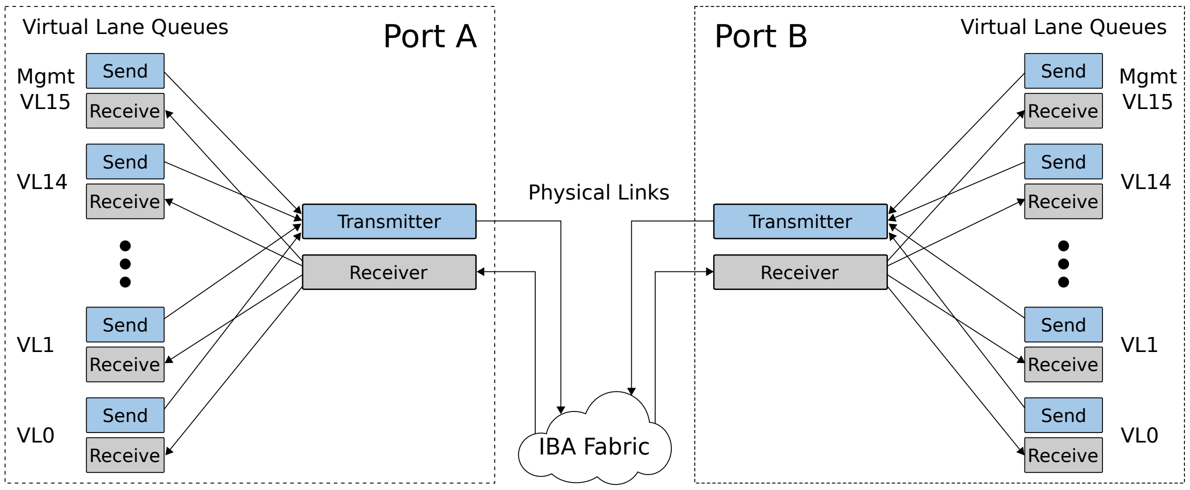


Figure 2.7: The virtual lanes in IBA

SL is mapped to a VL on each link based on the configured SL to VL mapping table on the port. IB supports up to 16 VLs. However, the last VL is reserved for subnet management traffic and cannot be used by user applications. The other 15 VLs are often referred to as data VLs.

2.4.6 The OFED Software Stack

The Open Fabrics Enterprise Distribution (OFED), developed and maintained by the OpenFabrics Alliance, is the de facto standard software stack for building and deploying RDMA and kernel bypass applications [84]. The OFED software package supports IBA, iWARP for Ethernet, and RDMA over Converged Ethernet (RoCE), and is available as open source for many Linux and Windows distributions, including RedHat Enterprise Linux, SUSE Linux Enterprise Distribution, Oracle Enterprise Linux (OEL) and Windows Server OSs. The OFED stack supports a variety of IBA adapters and switches, and includes drivers for virtually all major IB products. The subnet manger for IB bundled with the OFED is called *OpenSM*. We use OpenSM and the rest of the OFED stack for the development and demonstration of our prototypes throughout the work carried out in connection with this thesis.

2.5 Self-Adaptive Systems

Self-adaptivity can be defined as the property of a system to autonomously evaluate and change its behavior in response to a changing environment [185]. Self-adaptive systems play an important role in dynamic environments such as in situations where context awareness or interactivity is important. The application areas of such systems include, but are not limited to, embedded computing, mobile computing, Human-Computer interaction, sensor networks, *Service-oriented computing* [186], and IoT.

A common approach to organize self-adaptive systems is through a *feedback loop* [187], sometimes also referred to as *autonomic control loop* in the context of *autonomic computing* [188]. The main objective of self-adaptation is to empower computing systems with autonomic properties, often denoted as the *self-** properties. These properties include *self-*

configuration, *self-healing*, *self-protection*, and *self-optimization*, among others. In addition, a prerequisite of implementing the autonomic behavior in a system is that it should have *self-awareness* and *context-awareness*. That is, the system should have adequate mechanisms to gain knowledge about its internal states and the external operating environment [189].

Feedback loops are often implemented by a *Monitor-Analyze-Plan-Execute* (MAPE[11]) based adaptation sequence. In MAPE-based systems, the system being adapted, also called a *managed system*, is continuously monitored and analyzed, and an adaptation strategy is planned and executed, when a symptom stopping the system to achieve the desired behavior is detected [190–193]. Based on the adaptation mechanism, self-adaptive systems can be broadly classified into two categories: systems based on an internal control mechanism and systems using an external control mechanism. In the case of an internal control mechanism, the adaptation logic is tightly coupled with the application business logic. The adaptation is usually achieved by providing adaptation actions based on the values of internal application states. On the other hand, systems based on external control mechanisms provide a well-defined interface between the adaptation engine and the managed system. The managed system provides *monitors* and *effectors* to monitor the behavior of the system and take adaptation actions, respectively. The adaptation engine holds the adaptation logic, considers monitored parameters as an input, and decides when a particular effector should be triggered.

[11]MAPE is also referred to as MAPE-K, with K representing the shared knowledge-base required to implement all stages of the monitor, plan, and execute sequence [190].

Chapter 3

Contributions and Summary of Research Papers

In this chapter, the main contributions of this thesis are summarized. As most of the work conducted as part of this thesis is presented as a collection of research papers published in respected journals and conferences, we also provide extended abstracts of all these papers towards the end of this chapter.

3.1 Contributions

Our contributions lie within three main categories. First, we propose a set of improvements to the fat-tree routing algorithm to make it suitable for HPC workloads in the cloud. Our proposed improvements to the fat-tree routing make it more efficient, provides performance isolation among tenants in multi-tenant systems, and enable routing of both physical end nodes and virtualized end nodes according to the policies set by the provider. Second, we design new network reconfiguration methods to significantly reduce the time it takes to reroute the IB network. Reduced network reconfiguration time means that the interconnection network in a HPC cloud can optimize itself quickly to adapt to changing tenant configurations, SLAs, faults, running workloads, and current network conditions. Last, we successfully demonstrate a self-adaptive network prototype for IB-based HPC clouds, fully equipped with autonomous monitoring and adaptation, and configurable through a high-level *condition-action* language for the service providers. We now briefly summarize the contributions in each of these directions.

3.1.1 Improvements to the Fat-Tree Routing

The fat-tree routing algorithm [86, 87], as implemented in OpenSM, has several weaknesses that affect its performance and limit its suitability for a dynamic multi-tenant HPC cloud. We start from the load-balancing technique employed by the fat-tree routing algorithm, which is node-oblivious, and hence, is subject to issues leading to low network utilization and predictability. Addressing our first research question, *RQ1*, in Paper I [147] we proposed a *weighted* fat-tree routing algorithm (wFatTree) based on the original fat-tree routing algorithm. The wFatTree routing introduces the notion of *weights* assigned to each end node in the system. The routing algorithm considers the different weights assigned to the nodes and balances the network traffic more efficiently increasing overall network utilization. The weights can be static, based on the known characteristics of the end nodes such as their roles in the cluster or how critical the traffic flowing towards them is, or dynamic, based on learned traffic characteristics in the network as demonstrated later in the self-adaptive network prototype of Paper VII [153]. In addition to the efficient load-balancing, the wFat-Tree routing algorithm also fixes routing order issue of the fat-tree routing algorithm, which leads to unpredictability as the routing indexing order depends on the port numbers to which the nodes are attached – a configuration hard to control in dynamic HPC systems. Even though the weighted fat-tree routing we demonstrated in Paper I was specific to the fat-tree topologies, similar approaches can also be used in other topologies as well. We particularly note that a somewhat similar approach is later used by [79] to demonstrate *workload-aware* routing with the DFSSSP routing algorithm [76] in a topology-agnostic way.

Another important issue in HPC clouds is performance isolation among tenants, which is the subject of our research question *RQ2*. The original fat-tree routing algorithm, as well as other routing algorithms implemented in OpenSM, does not consider partitioning information related to the tenants in the subnet. Such tenant-oblivious routing leads to interference between tenant traffic of different partitions, and thus, results in unpredictable network performance. In Paper II [148][1], we proposed another set of improvements to fat-tree routing. The partition-aware routing algorithm (pFTree) improves performance isolation among tenant nodes. The algorithm employs both a physical link isolation mechanism and virtual lanes to reduce the impact of inter-partition interference in HPC networks, while maintaining high network utilization at the same time.

The pFTree routing algorithm presented in Paper II is unable to take different requirements and isolation criteria for different partitions. Not all tenants are subjected to the same SLAs in an HPC cloud, and thus, tenant nodes are to be assigned different isolation requirements in the network. Moreover, the weighted fat-tree routing and partition-aware routing algorithm, being subjected to different routing criteria, can lead to contradictory routing assignments. The Paper III [149] attack both these issues. We provide an extended version of the pFTree routing algorithm that takes provider-defined tenant-wise isolation policies into consideration when routing the fat-tree topology. In the second extension, we combine weighted fat-tree routing and partition-aware routing in an efficient and configurable manner. These contributions partly address our research question *RQ3*, which later gets further addressed in Paper VII.

Most cloud providers employ virtualization techniques [102], in terms of VMs, for flexible workload management in the infrastructure. The current shared-port SR-IOV architecture used in IB networks does not allow to route traffic destined to VMs independently of the physical machines where the VMs are being hosted. In this way, all VMs running on a physical machine are assigned the same paths in the topology, even when multiple paths (up to the physical machine) are available to be utilized, for segregating traffic towards colocated VMs belonging to different tenants, for instance. To cater this, a vSwitch SR-IOV architecture [104] can be employed. However, new routing strategies are required to

[1]Paper II was among the top 5 papers of IEEE/ACM CCGrid 2015 conference selected to be published as extended versions in a FGCS special edition, *Leading-edge research in cluster, cloud, and grid computing: Best papers from the IEEE/ACM CCGrid 2015 conference* [194].

efficiently route virtualized subnets, as indicated in *RQ4*. In Paper VI [152], we proposed and evaluated a fat-tree routing algorithm suitable for virtualized IB subnets based on the vSwitch architecture[2].

3.1.2 Efficient Network Reconfiguration Mechanisms

In large and dynamic HPC systems, such as HPC clouds, network reconfiguration is frequently needed to ensure connectivity and maintain optimal performance [107]. Reconfigurations in IB networks, however, are very costly. In general, on the reception of a reconfiguration event, such as a topology change, component failure, or required performance optimization, the installed routing function needs to be updated to cope with the change. The cost of reconfiguration, depending on the reconfiguration event, largely comes from two factors: the time required to reroute the network (LFT recalculations), and once new routing tables are calculated, the cost of sending SMPs to the switches containing LFT block updates. The cost related to the corresponding number of dropped packets during the reconfiguration comes in addition and depends on the employed reconfiguration model [195–197]. To address the reconfiguration requirements stemming from the dynamic nature of the clouds, as mentioned in our fifth research question, *RQ5*, we developed and gradually improved our reconfiguration mechanisms to reduce the cost of reconfiguration in fat-tree topologies. However, as with the other solutions presented in this thesis, many of the reconfiguration concepts developed can be easily applied to other topologies as well.

In Paper V [150], the concept of a *minimal routing update* (MRU) technique is presented. The basic idea of the MRU is that, given a set of end nodes in a topology, multiple routing functions with the same performance characteristics can be generated. On a reconfiguration event, generating the new routing function in a way that makes it the *closest* one to the existing configuration in the network, reduces the number of modifications needed to the existing paths substantially. In this way the routing function, to the extent possible, preserves existing forwarding entries in the switches reducing reconfiguration cost. Based on the MRU technique, we also proposed the *SlimUpdate* fat-tree routing algorithm in Paper V.

A challenge with the SlimUpdate routing algorithm is that, as the routing becomes complex considering the existing network configuration, the algorithm's execution time increases significantly. For performance-driven reconfiguration, however, it is only the assignments of the paths to the end nodes in the network that changes between reconfigurations. As the network topology remains the same, the existing paths in the network can actually be calculated only once. This is the idea behind our proposed metabase-aided re-routing method presented in Paper V [151]. In the proposed metabase-aided network reconfiguration method, routing is divided into two distinct phases: calculation of the paths in the topology, and the assignment of the calculated paths to the actual destinations. When a reconfiguration event requires no topology change, the path calculation phase can be completely eliminated by using a *metabase* with stored paths from the first phase, thus, significantly reducing network reconfiguration costs.

[2]Paper VI also presents minimum overhead reconfiguration on VM live migrations. However, the author's contributions as the second author of the paper were mainly directed towards routing and its evaluation.

3.1.3 A Self-Adaptive Network for IB

Our third area of contributions presents a modular framework to *stitch together* contributions from the other two areas by providing a self-adaptive provider-configurable network architecture for IB subnets. In dynamic HPC clouds, owing to multi-tenancy, rapid elasticity, and on-demand resource provisioning, static network reconfiguration is not feasible. Dynamic network optimization, irrespective of the optimization criteria employed, requires mechanisms to continuously monitor the system, and optimize it according to the changing conditions. In the context of HPC clouds, it is important that the load of the network links, the tenant node assignments, virtualization information, and current resource allocations are monitored continuously, and that the routing function is optimized based on the monitored metrics as well as provider-specific goals and requirements.

In Paper VII [153], we continue to explore *RQ3*, in addition to addressing *RQ6*. The paper deals with self-adaptation and maximization of the utility as perceived by the provider, and demonstrates a working self-adaptive IB network prototype for HPC clouds. Our proposed self-adaptive network framework is based on a feedback-driven control loop, powered by a monitoring service and a tailored rule-based *condition-action* language, *IBAdapt*, for defining provider-specific adaptation strategies. The optimization engine takes network and cloud metrics, gathered by the monitoring service, in account together with the provider-defined constraints and adaptation policies, and keep the IB network optimized without the need of any manual management interaction. The IBAdapt language itself is designed to evaluate a set of condition-action rules, and execute a predefined action specifying how the routing system should be modified in response to a specific condition. Apart from configurable system parameters, the IBAdapt rules can also be based on monitored metrics. For instance, a rerouting engine based on traffic profiles may be executed when the average throughput per node is decreased below a certain threshold. Practically, the adaptation engine can be used by any IB-based system, provided that the conditions and logic specifying what actions are needed is provided. Running as an autonomous system, a self-adaptive IB network helps achieving improved system performance and QoS compliance for the applications. Thus the net result of utilizing our proposed framework is an efficient and more predictable high performance cloud infrastructure with less SLA violations, well-suited to run both HPC and HPDA workloads.

3.2 Summary of Research Papers

In total, seven research papers constitute this thesis. In the following, we provide a short summary of each of these papers.

3.2.1 Paper I: A Weighted Fat-Tree Routing Algorithm for Efficient Load-Balancing in InfiniBand Enterprise Clusters

In Paper I [147], we studied the impact of node-oblivious load-balancing on the network performance and predictability in fat-tree topologies. The fat-tree routing algorithm [86,87], as currently implemented in the OpenSM, aims to generate LFTs that evenly spread routes across the links in the network fabric. The load-balancing technique employed by the fat-tree routing relies on a method that maintains a port usage counter, as described in Section 2.2.2, to choose the least-used port every time a new route is assigned to a link at a switch. The algorithm traverses the topology in the *indexing order* and assigns target LIDs of the end nodes, and thus the corresponding routes, to each switch port – while increasing the counter on every route assignment. However, one problem with this load-balancing technique is that it assumes a uniform traffic distribution in the network. When routes towards nodes that mainly consume large amount of data are assigned to share links in the fabric while alternative links are underutilized, sub-optimal network throughput is obtained. Also, as the fat-tree algorithm routes nodes according to the *indexing order*, the performance may differ for two systems cabled in the exact same way. For the nodes connected to the same leaf switch, the indexing order depends on the switch port number to which a node is connected, which is hard to maintain.

In this paper we proposed wFatTree, a novel and efficient weighted fat-tree routing algorithm that improves actual load-balancing in the network. The algorithm exploits known or learned node traffic characteristics and generates efficient routing tables to evenly balance load across the links in the network. At the same time, as the algorithm relies on a deterministic indexing order, predictable network performance is achieved. Furthermore, wFatTree can also be used to prioritize traffic flowing towards critical nodes in the network. Our experiments and simulations, based on the wFatTree implementation in OpenSM, show up to 60% improvement in total network throughput over the original fat-tree routing algorithm for large fat-trees.

3.2.2 Paper II: Partition-Aware Routing to Improve Network Isolation in InfiniBand Based Multi-tenant Clusters

In IB subnets, isolation of nodes is provided through partitioning. Partitioning defines a mechanism for creating logical groups of ports such that the communication is restricted only to the members of a group. At host channel adapters and switches, packets are filtered using the partition membership information to enforce isolation. The routing algorithm, however, is unaware of these partitions in the network. As a result, traffic flows belonging to different tenant partitions might share links inside the network fabric. This sharing of intermediate links creates interference, which is particularly critical to avoid in multi-tenant environments, like HPC clouds. Ideally, each tenant should receive a predictable network performance, unaffected by the workload of the other tenants in the system. In addition, using the current routing schemes, despite the fact that links connecting nodes outside partitions are never used, they are routed the same way as the other functional links. This may result in degraded load-balancing.

In Paper II [148], we proposed a partition-aware fat-tree routing algorithm, pFTree. The pFTree routing algorithm utilizes a multifold mechanism to provide performance isolation among partitions belonging to different tenant groups in a subnet. At the same time, it generates well-balanced LFTs for fat-tree topologies by distributing routes evenly across the links in the tree. The pFTree uses partitioning information about the subnet to remove contention between paths belonging to different tenant groups. Given the available network resources, pFTree starts isolating partitions at the physical link level. If the topology does not have enough network resources to provide complete physical partition isolation at each topology level without compromising on the load-balancing, the pFTree employs a complementary virtual lane based isolation scheme that works in conjunction with the physical isolation to reduce the impact of contention. Using both lab experiments and simulations, we showed that pFTree is able to significantly reduce the effect of inter-partition interference effectively without any additional functional overhead. Furthermore, pFTree also provides improved load-balancing over the state-of-the-art fat-tree routing algorithm.

3.2.3 Paper III: Efficient network isolation and load balancing in multi-tenant HPC clusters

Building on the partition-aware routing introduced in Paper II [148], in Paper III [149] we provided two significant extensions to the pFTree routing algorithm for fat-tree topologies. In the first extension, we improved the pFTree routing algorithm to include partition-wise isolation policies. For each partition, the isolation policies determine how the nodes in that partition are allowed to share resources with nodes belonging to the other partitions in the system. As mentioned above, when the network does not have enough resources to isolate partitions solely at the physical link level, the pFTree routing algorithm uses VLs to reduce inter-partition interference. However, different partitions may have different isolation needs depending on the corresponding SLAs or QoS requirements. For example, some of the partitions in the network may be running critical operations, and may require complete physical isolation in all cases. Similarly, in many networks, depending on the availability of the VLs, some partitions may have to share a VL with another partition, which may not be desirable for communication-intensive workloads. The extended pFTree algorithm, with the help of the isolation policies, is able to specify the aforementioned partition-wise requirements in the routing, and thus the partitions are treated according to their QoS requirements.

In the second extension to the pFTree routing presented in Paper III, a weighted version of the pFTree routing algorithm was presented. The weighted pFTree routing algorithm, besides partitions, also takes node's traffic characteristics into account to balance load across the network links more efficiently. A comprehensive evaluation of both extensions, comprising hardware experiments and simulations, confirmed the correctness and feasibility of the proposed extensions.

3.2.4 Paper IV: SlimUpdate: Minimal Routing Update for Performance Based Reconfigurations in Fat-Trees

In Paper IV [150], we attacked the issue of costly network reconfigurations in IB networks. Dynamic network reconfiguration in statically routed IB networks requires computation and distribution of a new set of routes to the switches, implemented by means of the LFTs. Whenever a network reconfiguration event is triggered, the OpenSM employs a routing algorithm to compute new LFTs with the updated topology information, in order to maintain connectivity and performance. In general, current routing algorithms do not consider the existing routes in a network when calculating new ones. Such configuration-oblivious routing might result in substantial modifications to the existing paths, and the reconfiguration becomes costly as it potentially involves a large number of source-destination pairs. For these reasons, dynamic reconfiguration in IB is mainly restricted to fault-tolerance, and performance-based reconfigurations are not well-supported.

In this paper, we proposed the MRU scheme as described in Section 3.1.2. The MRU reuse the existing paths in the network while conducting network reconfigurations to reduce configuration costs. We also presented a fat-tree routing algorithm, SlimUpdate, which utilizes the proposed MRU technique. Our experiments and simulations show a decrease of up to 80% in the number of total path modifications when using SlimUpdate routing, while achieving similar or even better performance than the fat-tree routing in most reconfiguration scenarios. A challenge remained, however, is that the SlimUpdate routing algorithm is complex, requiring some additional computational overhead compared to the de facto fat-tree routing. The exact SlimUpdate routing time during the reconfiguration phase depends specifically on the reconfiguration scenario and the complexity of the reconfiguration event. For performance related reconfigurations, however, SlimUpdate is quite beneficial as the routing can be calculated *off-line* without disturbing the current configuration.

3.2.5 Paper V: Compact network reconfiguration in fat-trees

Paper V [151] is an extended version of Paper IV [150]. In this paper, we introduced a new metabase-aided network reconfiguration method as briefed in Section 3.1.2. The implementation of the metabase-aided two phase routing for fat-tree topologies is based on destination leaf-switch multipathing. In a fat-tree topology, assuming nodes are *single-homed*, each node port is connected to a single leaf-switch. This implies that the path between any source–destination pair, as calculated by a fat-tree routing algorithm, corresponds to one of the available multiple leaf-switch to leaf-switch paths between corresponding leaf switches. In our proposed two-phase destination-based routing scheme, paths towards leaf switches are calculated using multipath routing in the first routing phase. The path calculation is done regardless of the end nodes actually connected to the leaf switches. In the second phase, calculated paths are allocated to the end nodes, and LFTs are generated accordingly. The calculated multipath routing blueprint from the first phase, is stored in a metabase and used later for fast network reconfigurations. When a performance-driven reconfiguration is triggered, without a topology change, the routing algorithm can simply re-use the calculated paths from the metabase and assign them to the compute nodes according to its path assignment logic. For instance, paths can be assigned to the compute nodes based on their current network profiles, or in correspondence with a given priority criteria.

The proposed metabase-aided reconfiguration method significantly reduces network reconfiguration overhead, while providing greater routing flexibility. On successive runs, our proposed method saves up to 85% of the total routing time over the traditional rerouting schemes. Based on the metabase-aided routing, we also presented an updated SlimUpdate routing algorithm that dynamically optimizes routes for a given MPI node order. The modified SlimUpdate routing algorithm is able to reach the peak bandwidth for a specific communication pattern taking the node order of the MPI communicator into consideration when routing the topology.

3.2.6 Paper VI: Efficient Routing and Reconfiguration in Virtualized HPC Environments with vSwitch-enabled Lossless Networks

To meet the demands of communication-intensive workloads in the cloud, VMs should utilize low overhead network communication paradigms. In general, such paradigms enable VMs to directly communicate with the hardware by means of a passthrough technology like SR-IOV. However, when passthrough-based virtualization is coupled with lossless interconnection networks, live-migrations introduce scalability challenges due to the substantial network reconfiguration overhead. With these challenges in mind the vSwitch architecture is proposed [104]. In Paper VI [152], solutions to rectify the scalability issues present in vSwitch-enabled IB subnets are presented, together with routing strategies for virtualized fat-tree topologies based on the vSwitch implementation.

To obtain optimal performance, the routing algorithm should consider the vSwitch architecture when calculating routes. The vSwitches can be identified in the topology discovery process by the distinct property of having only one upward link to the corresponding switch. Once the vSwitches have been identified, a routing function can generate LFTs for all the switches such that the traffic can find its path towards all VMs in the network. In the proposed vSwitch architecture each VM has its own address, thus, technically each VM can be routed independently of other VMs attached to the same vSwitch. One drawback of this approach is that when the VM distribution is not uniform among vSwitches, the vSwitches with more VMs are potentially assigned more network resources, irrespective of the actual traffic distribution. In this paper, we proposed a weighted routing scheme for vSwitch-based virtualized subnets. In this scheme, the cumulative weight of VMs per vSwitch will be equal on all vSwitches, so the links in the topology can be balanced without being affected by the actual VM distribution. At the same time, the strategy enables multipath routing where each VM can be independently routed in the network, eliminating interference between same vSwitch VMs at the intermediate links in the topology. The scheme can be combined with per VM rate limit enforcement on each vSwitch to ensure that a VM is not allowed to exceed its allocated capacity. In addition, in the presence of multiple tenant groups in the network, techniques like pFTree routing can also be integrated to provide network-wide isolation among tenants. We implemented the routing algorithm for vSwitch-enabled fat-trees and evaluated its performance. As vSwitch-enabled hardware does not yet exist, evaluation was done by emulating vSwitches with existing hardware, complemented by simulations.

3.2.7 Paper VII: A Self-Adaptive Network for HPC Clouds: Architecture, Framework, and Implementation

After implementing solutions tackling different challenges with respect to HPC clouds, in Paper VI [153], we provide a framework for implementing a self-adaptive IB network where our prototypes can be readily combined together in a single holistic and autonomous solution.

The self-adaptive framework proposed in this paper consists of two layers: a system layer and an adaptation layer. The system layer is concerned with the underlying managed system, that in our case consists of an IB fabric, while the adaptation layer provides capabilities to implement the self-adaptation. Our solution, based on a feedback-driven control and optimization loop, enables an IB network to dynamically adapt to varying traffic patterns, current resource availability, and workload distributions, and all in accordance with service provider-defined policies. Furthermore, we present IBAdapt, which is a rule-based language (as discussed in Section 3.1.3) for the service providers to specify adaptation strategies used by the framework. Results obtained on a test cluster demonstrate the feasibility and effectiveness of the framework implementation in improving Quality-of-Service compliance in clouds.

Chapter 4

Closing Remarks and Future Work

All of the research questions addressed in this thesis, as listed in Section 1.2, are linked to one fundamental research problem: *How can we efficiently realize HPC clouds*? Previous research have shown that the network costs can account for up to 33% of the total system costs [198]. In addition, overall energy consumption in HPC systems is largely attributed to interconnection networks, with estimates suggesting that the network use as much as 50% of the total energy in large HPC installations [199]. In this connection, given a fixed topology and switching layer techniques, we identify routing as a fundamental property affecting both the costs and the performance of the interconnection networks. In particular, current state-of-the-art routing schemes are not suitable for dynamic HPC clouds, and hence need to be optimized taking cloud-specific requirements into consideration.

The research conducted in this thesis has potential impacts on both private cloud infrastructures, such as medium sized clusters used for enterprise HPC, and public clouds offering innovative HPC solutions to the customers at scale. Moreover, as our work was conducted in close collaboration with the industrial partner Oracle, future Oracle products may include research solutions developed as part of this thesis. The industrial application of the thesis is reflected by the eight corresponding patent applications listed in Section 1.5.

Throughout this work, we have used IB as the interconnect technology to demonstrate and prototype the methods, solutions, and techniques designed and developed; and different variants of the fat-tree topology for evaluation purposes. The selection of IB and the fat-tree topology is largely attributed to their efficiency and popularity in HPC systems, as already justified in Chapter 1. In this area, apart from the exploitation by our industry collaborators, our contributions have already started gaining some interest in the general HPC and IB community [200]. However, most of the conceptual solutions to the problems addressed in this work are equally applicable to other lossless interconnection networks and topologies. In this context, some of the research results from this thesis will continue to be exploited in the coming years. For example, some of our results are planned to be incorporated in a closely-linked European research and innovation project, Melodic. The basic vision of Melodic is to enable federated and distributed multi-cloud computing for data-intensive applications. The concepts from this thesis related to network optimizations for enabling high-performance applications and data analytics are also beneficial for Melodic, albeit, with a different set of technologies and challenges.

Future Work

Several directions for future work can be identified from our contributions as presented in Chapter 3. Broadly speaking, the future work can be categorized into two major classes. First, improvements and further work on the proposed routing algorithms, network reconfiguration techniques, and self-adaptive network solution is possible. Second, adaptation of the presented solutions and techniques to other topologies and interconnection networks can help to generalize the solutions and their exploitation in a variety of systems.

A limitation of the weighted fat-tree routing algorithm, as presented in Paper I [147], is that it does not consider different link capacities that may be available in a topology. When all links are not operating at the same speed, effective throughput received by a traffic flow is determined by the bottleneck link capacity. The wFTree routing algorithm can be updated to include consideration of the link capacities when assigning paths to different links in the network. An important challenge, however, is maintaining efficiency of the algorithm as considering available link capacities for different flows, while assigning routes, may be computationally expensive increasing the routing time.

As mentioned in Paper II [148], the IB architecture supports adding a port to multiple partitions. Also, two partition membership types are supported, *limited* and *full* membership. Limited members are restricted to communicate only with the full members of the partition. As per the current implementation, the pFTree routing algorithm, as well as its extended versions from Paper III [149], does not support overlapping partitions and limited membership types. Although a node is allowed to be a member of multiple partitions, only one of those partitions can be marked as primary, to be used in pFTree routing. One way to handle this issue is to allow different priorities for the different partitions on the same node. Also, the algorithm can be extended to use multi-path routing where different partitions for a single node can be mapped to different routes, and have different isolation characteristics. Furthermore, the notion of *partition groups* can be added to the algorithm, prescribing which specific partitions form a single group of nodes communicating with each other. This can be useful in HPC clouds where some of the tenant nodes communicate with nodes belonging to other tenants in the system.

The network reconfiguration techniques presented in Paper IV [150], and Paper V [151] substantially reduce network reconfiguration overhead in the performance-driven reconfiguration scenarios. For performance-driven reconfiguration, it is safe to assume that the topology is not changed in between the reconfigurations. However, the same approach can be also be applied for fault-tolerance. In particular, for the metabase-aided reconfiguration method presented in Paper V [151], the routing algorithm can *pre-calculate* a number of alternative paths that can be quickly assigned to fix disconnected routing in case of a failure in the network. Such an approach can improve the routing efficiency in case of failures in large HPC systems, and can be beneficial to the fail-in-place network design as well [143].

In Paper VI [152], we proposed routing strategies for virtualized IB subnets implementing a *vSwitch*-based virtualization model. The virtualization approach, used by the IB standard, has since been standardized to include the concept of a *vPort*. The vPort model can be thought of as a hybrid approach possessing characteristics from both the *shared-port* and the *vSwitch* models. Specifically, a vPort has its own *GID*, partitioning membership, and dedicated links states. Moreover, optionally a unique *LID* can also be associated with a vPort [105, 106]. The subnet manager manages vPorts similar to physical ports, and hence, the routing function can take both the vPorts and the physical ports into account to utilize the topology in the most efficient way. As the vPort virtualization is now the standard adopted by the IB specification, routing strategies for virtualized subnets should be revisited. Recently, a VM migration framework for SR-IOV enabled IB networks is also presented [201], which can be evaluated for inclusion in the devised virtualization architecture in Paper VI [152].

The self-adaptive network framework, presented in Paper VII [153], can be thought of as a *proof-of-concept* with several opportunities available for improvements. As mentioned in Paper VII, the adaptation framework and the IBAdapt language can be extended to include support for more flexible adaptation policies, SLAs, and invariants used by the adaptation engine. In addition, theoretical self-adaptation approaches can be utilized in the adaptation engine, such as the support of context-aware QoS dimensions and various state-of-the-art utility-based approaches for decision-making. A *model@runtime* [202] based system can also be beneficial as a large amount of work has already been done in this direction in different domains [203–205].

Adapting the work presented in this thesis to other interconnection networks, topologies, and routing algorithms is also an interesting future direction to explore. In particular, the problems hindering the realization of efficient HPC clouds are neither specific to IB nor to the use of the fat-tree topology in such systems. However, different technologies bring their own subset of challenges as well as opportunities, and our solutions may need tailoring according to the application area. For instance, the facility of *dispersive routing*, which allows a source node to spread traffic across multiple available paths can complement the weighted fat-tree routing in an Omni-Path architecture [52]. In the context of HPC solutions employing other topologies, an important consideration is the possibility of deadlocks[1] during routing and network reconfiguration. Lossless interconnection networks are particularly prone to deadlocks as packet drops are not allowed during regular operation. Deadlock freedom is an aspect that we do not particularly focused on in our work because the use of the fat-tree topology makes routing possible without special considerations with regards to deadlock avoidance.

[1]A deadlock is a situation where packets cannot progress because of cyclic resource dependencies among them. That is, the packets are holding on some resources while waiting to acquire others, resulting in a circular chain of dependencies [206].

Bibliography

[1] Top 500 Super Computer Sites. http://www.top500.org/, accessed July 1, 2017.

[2] Jack R Collins, Robert M Stephens, Bert Gold, Bill Long, Michael Dean, and Stanley K Burt. An exhaustive DNA micro-satellite map of the human genome using high performance computing. *Genomics*, 82(1):10–19, 2003.

[3] Xiuwen Zheng, David Levine, Jess Shen, Stephanie M Gogarten, Cathy Laurie, and Bruce S Weir. A high-performance computing toolset for relatedness and principal component analysis of SNP data. *Bioinformatics*, 28(24):3326–3328, 2012.

[4] J Michalakes, J Dudhia, D Gill, T Henderson, J Klemp, W Skamarock, and W Wang. The weather research and forecast model: software architecture and performance. In *Proceedings of the Eleventh ECMWF Workshop on the Use of High Performance Computing in Meteorology*, pages 156–168. World Scientific: Singapore, 2005.

[5] KY Sanbonmatsu and C-S Tung. High performance computing in biology: multimillion atom simulations of nanoscale systems. *Journal of structural biology*, 157(3):470–480, 2007.

[6] Tim McGuire, James Manyika, and Michael Chui. Why big data is the new competitive advantage. *Ivey Business Journal*, 76(4):1–4, 2012.

[7] Chris Yiu. The big data opportunity. *Policy Exchange*, pages 1–36, 2012.

[8] Andrea De Mauro, Marco Greco, Michele Grimaldi, Georgios Giannakopoulos, Damianos P Sakas, and Daphne Kyriaki-Manessi. What is big data? A consensual definition and a review of key research topics. In *AIP conference proceedings*, volume 1644, pages 97–104. AIP, 2015.

[9] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, 36(4):1165–1188, 2012.

[10] Ritu Arora. Conquering big data with high performance computing, 2016.

[11] Doug Black. IDC: AI, HPDA Driving HPC into High Growth Markets. https://www.hpcwire.com/2016/11/16/idc-ai-hpda-driving-hpc-high-growth-markets/, accessed July 1, 2017.

[12] Geoffrey Fox, Judy Qiu, Shantenu Jha, Saliya Ekanayake, and Supun Kamburugamuve. Big data, simulations and hpc convergence. In *Workshop on Big Data Benchmarks*, pages 3–17. Springer, 2015.

[13] Peter Mell and Tim Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50, 2009.

[14] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[15] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. Cloud computing: An overview. *Cloud computing*, pages 626–631, 2009.

[16] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[17] Robert M Metcalfe and David R Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, 1976.

[18] Jeffrey Napper and Paolo Bientinesi. Can cloud computing reach the top500? In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, pages 17–20. ACM, 2009.

[19] Michal Husejko, Ioannis Agtzidis, Pierre Baehler, Tadeusz Dul, John Evans, Nils Himyr, and Helge Meinhard. HPC in a HEP lab: lessons learned from setting up cost-effective HPC clusters. In *Journal of Physics: Conference Series*, volume 664, page 092012. IOP Publishing, 2015.

[20] InfiniBand Architecture Specification: Release 1.3, 2016. http://www.infinibandta.com/, accessed July 1, 2017.

[21] Vernon Turner, John F Gantz, David Reinsel, and Stephen Minton. The digital universe of opportunities: Rich data and the increasing value of the internet of things. *IDC Analyze the Future*, 2014.

[22] IDC: The premier global market intelligence firm. https://www.idc.com/, accessed July 1, 2017.

[23] Iot Platforms: Enabling the Internet of Things, 2016. https://www.ihs.com/Info/0416/internet-of-things.html, accessed July 1, 2017.

[24] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[25] Paul DuBois. *MySQL cookbook.* " O'Reilly Media, Inc.", 2006.

[26] Bruce Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.

[27] Doug Laney. 3D data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6:70, 2001.

[28] Tatiana Lukoianova and Victoria L Rubin. Veracity roadmap: Is big data objective, truthful and credible? 2014.

[29] Ioanna D Constantiou and Jannis Kallinikos. New games, new rules: big data and the changing context of strategy. *Journal of Information Technology*, 30(1):44–57, 2015.

[30] Douglas F Parkhill. The Challenge of the computer utility. 1966.

[31] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. *INC, IMS and IDC*, pages 44–51, 2009.

[32] Kenji E Kushida, Jonathan Murray, and John Zysman. Cloud computing: from scarcity to abundance. *Journal of Industry, Competition and Trade*, 15(1):5–19, 2015.

[33] Michael A Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32, 2004.

[34] ETP4HPC. Strategic Research Agenda 2015 Update - European Technology Multi-annual Roadmap Towards Exascale, 2016. http://www.etp4hpc.eu/image/fotos/2016/01/ETP4HPC-SRA-2-Single-Page.pdf, accessed July 1, 2017.

[35] Cloud helps Elon Musk further his audacious goals, 2017. http://www.networkworld.com/article/3168927/cloud-computing/cloud-helps-elon-musk-further-his-audacious-goals.html, accessed July 1, 2017.

[36] Elon Musk. Hyperloop alpha. 2013. http://www.spacex.com, hyperloop_alpha-20130812.pdf, accessed July 1, 2017.

[37] Microsoft Azure Cloud Computing Platform and Services. https://azure.microsoft.com/, accessed July 1, 2017.

[38] Thomas Sterling and Dylan Stark. A high-performance computing forecast: Partly cloudy. *Computing in Science & Engineering*, 11(4):42–49, 2009.

[39] Qiming He, Shujia Zhou, Ben Kobler, Dan Duffy, and Tom McGlynn. Case study for running HPC applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 395–401. ACM, 2010.

[40] Abhishek Gupta and Dejan Milojicic. Evaluation of hpc applications on cloud. In *Sixth Open Cirrus Summit (OCS), 2011.*, pages 22–26. IEEE, 2011.

[41] Piyush Mehrotra, Jahed Djomehri, Steve Heistand, Robert Hood, Haoqiang Jin, Arthur Lazanoff, Subhash Saini, and Rupak Biswas. Performance evaluation of Amazon EC2 for NASA HPC applications. In *Proceedings of the 3rd workshop on Scientific Cloud Computing*, pages 41–50. ACM, 2012.

[42] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 159–168. IEEE, 2010.

[43] Intersect360 Research. http://www.intersect360.com/, accessed July 1, 2017.

[44] HPC Market Model and Forecast: 2016 to 2021 Snapshot Analysis, 2016. http://www.intersect360.com/industry/reports.php?id=148, accessed July 1, 2017.

[45] Bin Wang, Zhengwei Qi, Ruhui Ma, Haibing Guan, and Athanasios V Vasilakos. A survey on data center networking for cloud computing. *Computer Networks*, 91:528–547, 2015.

[46] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[47] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[48] Amazon Elastic Compute Cloud (Amazon EC2). https://aws.amazon.com/ec2/, accessed July 1, 2017.

[49] Paolo Bientinesi, Roman Iakymchuk, and Jeff Napper. HPC on competitive cloud resources. In *Handbook of Cloud Computing*, pages 493–516. Springer, 2010.

[50] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A performance analysis of EC2 cloud computing services for scientific computing. In *Cloud computing*, pages 115–131. Springer, 2009.

[51] Robert Alverson, Duncan Roweth, and Larry Kaplan. The gemini system interconnect. In *IEEE 18th Annual Symposium on High Performance Interconnects (HOTI)*, pages 83–87. IEEE, 2010.

[52] Mark S Birrittella, Mark Debbage, Ram Huggahalli, James Kunz, Tom Lovett, Todd Rimmer, Keith D Underwood, and Robert C Zak. Intel® Omni-path Architecture: Enabling Scalable, High Performance Fabrics. In *IEEE 23rd Annual Symposium on High-Performance Interconnects (HOTI)*, pages 1–9. IEEE, 2015.

[53] Fabrizio Petrini, Wu-chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics network: High-performance clustering technology. *IEEE Micro*, 22(1):46–57, 2002.

[54] Nanette J Boden, Danny Cohen, Robert E Felderman, Alan E. Kulawik, Charles L Seitz, Jakov N Seizovic, and Wen-King Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[55] William E Baker, Robert W Horst, David P Sonnier, and William J Watson. A flexible ServerNet-based fault-tolerant architecture. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing, (FTCS-25), Digest of Papers*, pages 2–11. IEEE, 1995.

[56] The InfiniBand Trade Association. http://www.infinibandta.org/, July 1, 2017.

[57] Viktor Mauch, Marcel Kunze, and Marius Hillenbrand. High performance cloud computing. *Future Generation Computer Systems*, 29(6):1408–1416, 2013.

[58] 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*, page 9. ACM, 2012.

[59] Paul Rad, Rajendra V Boppana, Palden Lama, Gilad Berman, and Mo Jamshidi. Low-latency software defined network for high performance clouds. In *10th System of Systems Engineering Conference (SoSE)*, pages 486–491. IEEE, 2015.

[60] Jie Zhang, Xiaoyi Lu, Mark Arnold, and Dhabaleswar K Panda. MVAPICH2 over OpenStack with SR-IOV: An Efficient Approach to Build HPC Clouds. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CC-Grid)*, pages 71–80. IEEE, 2015.

[61] Jie Zhang, Xiaoyi Lu, Sourav Chakraborty, and Dhabaleswar K DK Panda. Slurm-V: Extending Slurm for Building Efficient HPC Cloud with SR-IOV and IVShmem. In *European Conference on Parallel Processing (EuroPar)*, pages 349–362. Springer, 2016.

[62] OrionVM: Wholesale Cloud IaaS Provider. http://www.orionvm.com/, accessed July 1, 2017.

[63] ProfitBricks: Cloud Computing Infrastructure Services. https://www.profitbricks.com/, accessed July 1, 2017.

[64] Michael Feldman. Machine Learning is The Killer App for High Performance Computing. https://www.top500.org/news/machine-learning-is-the-killer-app-for-high-performance-computing/, accessed July 1, 2017.

[65] Tim o'Reilly. *What is web 2.0.* "O'Reilly Media, Inc.", 2009.

[66] Paul McNamara. What is Hyperscale and why is it so important to enterprises? http://cloudblog.ericsson.com/what-is-hyperscale-and-why-is-it-so-important-to-enterprises, accessed July 1, 2017.

[67] InfiniBand and Proprietary Networks Still Rule Real HPC. https://www.nextplatform.com/2017/06/30/infiniband-proprietary-networks-still-rule-real-hpc/, accessed July 1, 2017.

[68] Press Release: June 2017 Top500 List Shows InfiniBand Continues to be the Interconnect of Choice for High Performance Computing And Artificial Intelligence. http://www.infinibandta.org/content/pages.php?pg=press_room_item&rec_id=893, accessed July 1, 2017.

[69] Jose Duato, Sudhakar Yalamanchili, and Lionel M Ni. *Interconnection networks: An engineering approach*. Morgan Kaufmann, 2003.

[70] Radia Perlman. An algorithm for distributed computation of a spanning tree in an extended LAN. In *ACM SIGCOMM Computer Communication Review*, volume 15, pages 44–53. ACM, 1985.

[71] Charles E Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 100(10):892–901, 1985.

[72] German Rodriguez, Cyriel Minkenberg, Ramon Beivide, Ronald P Luijten, Jesus Labarta, and Mateo Valero. Oblivious routing schemes in extended generalized fat tree networks. In *IEEE International Conference on Cluster Computing and Workshops (CLUSTER)*, pages 1–8. IEEE, 2009.

[73] Bogdan Prisacari, German Rodriguez, Cyriel Minkenberg, and Torsten Hoefler. Fast pattern-specific routing for fat tree networks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4):36, 2013.

[74] Andres Mejia, Jose Flich, Jose Duato, S-A Reinemo, and Tor Skeie. Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori. In *20th International Parallel and Distributed Processing Symposium, (IPDPS)*, pages 10–pp. IEEE, 2006.

[75] William J Dally and Charles L Seitz. Deadlock-free message routing in multiprocessor interconnection networks. 1988.

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

[77] Michael D. Schroeder, Andrew D Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications.*, 9(8):1318–1335, 1991.

[78] Tor Skeie, Olav Lysne, and Ingebjørg Theiss. Layered Shortest Path (LASH) Routing in Irregular System Area Networks. In *International Parallel and Distributed Processing Symposium (IPDPS)*, volume 2, page 194. Citeseer, 2002.

[79] Jens Domke and Torsten Hoefler. Scheduling-aware routing for supercomputers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, page 13. IEEE Press, 2016.

[80] Michel A Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh, Marten Van Dijk, and Srinivas Devadas. *Application-aware deadlock-free oblivious routing*, volume 37. ACM, 2009.

[81] Vincenzo Catania, Rickard Holsmark, Shashi Kumar, and Maurizio Palesi. A methodology for design of application specific deadlock-free routing algorithms for NoC systems. In *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis, (CODES+ ISSS)*, pages 142–147. IEEE, 2006.

[82] Xiangke Liao, Liquan Xiao, Canqun Yang, and Yutong Lu. MilkyWay-2 supercomputer: system and application. *Frontiers of Computer Science*, 8(3):345–356, 2014.

[83] Haohuan Fu, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, Fangli Qiao, et al. The Sunway TaihuLight supercomputer: system and applications. *Science China Information Sciences*, 59(7):072001, 2016.

[84] The OpenFabrics Enterprise Distribution (OFED) Overview. https://www.openfabrics.org/index.php/openfabrics-software.html, accessed July 1, 2017.

[85] Herbert Sullivan and Theodore R Bashkow. A large scale, homogeneous, fully distributed parallel machine, I. In *ACM SIGARCH computer architecture news*, volume 5, pages 105–117. ACM, 1977.

[86] Eitan Zahavi, Gregory Johnson, Darren J Kerbyson, and Michael Lang. Optimized InfiniBand fat-tree routing for shift all-to-all communication patterns. *Concurrency and Computation: Practice and Experience*, 22(2):217–231, 2010.

[87] Eitan Zahavi. D-Mod-K routing providing non-blocking traffic for shift permutations on real life fat trees. *CCIT Report 776, Technion*, 2010.

[88] William James Dally and Brian Patrick Towles. *Principles and Practices of Interconnection Networks*. Elsevier, 2004.

[89] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. 2009.

[90] Shu-Ching Wang, Kuo-Qin Yan, Wen-Pin Liao, and Shun-Sheng Wang. Towards a load balancing in a three-level cloud computing network. In *3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, volume 1, pages 108–113. IEEE, 2010.

[91] Zehua Zhang and Xuejie Zhang. A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In *2nd International Conference on Industrial Mechatronics and Automation (ICIMA)*, volume 2, pages 240–243. IEEE, 2010.

[92] Xiaona Ren, Rongheng Lin, and Hua Zou. A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast. In *IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pages 220–224. IEEE, 2011.

[93] Ratan Mishra and Anant Jaiswal. Ant colony optimization: A solution of load balancing in cloud. *International Journal of Web & Semantic Technology*, 3(2):33, 2012.

[94] Juan Carlos Martinez, Jose Flich, Antonio Robles, Pedro Lopez, and Jose Duato. Supporting fully adaptive routing in infiniband networks. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 10–pp. IEEE, 2003.

[95] Zhu Ding, Raymond R Hoare, Alex K Jones, and Rami Melhem. Level-wise scheduling algorithm for fat tree interconnection networks. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 96. ACM, 2006.

[96] Diego Lugones, Daniel Franco, and Emilio Luque. Fast-Response Dynamic Routing Balancing for high-speed interconnection networks. In *IEEE International Conference on Cluster Computing and Workshops (CLUSTER)*, pages 1–9. IEEE, 2009.

[97] Diego Lugones, Daniel Franco, and Emilio Luque Fadón. Dynamic routing balancing on InfiniBand network. *Journal of Computer Science & Technology*, 8, 2008.

[98] Crispín Gomez, Francisco Gilabert, María Engracia Gomez, Pedro Lopez, and Jose Duato. Deterministic versus adaptive routing in fat-trees. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8. IEEE, 2007.

[99] Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. On the performance variability of production cloud services. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 104–113. IEEE, 2011.

[100] Ana Jokanovic, Jose Carlos Sancho, German Rodriguez, Alejandro Lucero, Cyriel Minkenberg, and Jesus Labarta. Quiet neighborhoods: Key to protect job performance predictability. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 449–459. IEEE, 2015.

[101] Eitan Zahavi, Alexander Shpiner, Ori Rottenstreich, Avinoam Kolodny, and Isaac Keslassy. Links as a Service (LaaS): Guaranteed tenant isolation in the shared cloud. In *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*, pages 87–98. ACM, 2016.

[102] Jyotiprakash Sahoo, Subasish Mohapatra, and Radha Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Second International Conference on Computer and Network Technology (ICCNT)*, pages 222–226. IEEE, 2010.

[103] Patrick Kutch. PCI-SIG SR-IOV primer: An introduction to SR-IOV technology. *Intel application note*, pages 321211–002, 2011.

[104] Evangelos Tasoulas, Ernst Gunnar Gran, Bjørn Dag Johnsen, Kyrre Begnum, and Tor Skeie. Towards the InfiniBand SR-IOV vSwitch Architecture. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 371–380. IEEE, 2015.

[105] InfiniBand Architecture Specification: Annex A 18: Virtualization, 2016. https://cw.infinibandta.org/document/dl/8126, accessed July 1, 2017.

[106] Liran Liss. InfiniBand Virtualization. In *13th Annual OpenFabrics Alliance Workshop.* OpenFabrics Alliance, 2017.

[107] Bianca Schroeder and Garth A Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350, 2010.

[108] Andreas Berl, Erol Gelenbe, Marco Di Girolamo, Giovanni Giuliani, Hermann De Meer, Minh Quan Dang, and Kostas Pentikousis. Energy-efficient cloud computing. *The Computer Journal*, 53(7):1045–1051, 2010.

[109] Fernando Guimaraes. *Research: Anyone can do it.* Pedia Press, 2003.

[110] Victor R Basili. Software development: A paradigm for the future. In *Proceedings of the 13th Annual International Computer Software and Applications Conference (COMPSAC)*, pages 471–485. IEEE, 1989.

[111] Gordana Dodig-Crnkovic. Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*, pages 126–130, 2002.

[112] Victor Basili. The experimental paradigm in software engineering. *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 1–12, 1993.

[113] Peter J Denning. ACM President's Letter: What is experimental computer science? *Communications of the ACM*, 23(10):543–544, 1980.

[114] Robert B Cooper. *Introduction to queueing theory.* North Holland,, 1981.

[115] Arnold O Allen. *Probability, statistics, and queueing theory.* Academic Press, 2014.

[116] C Newell. *Applications of queueing theory*, volume 4. Springer Science & Business Media, 2013.

[117] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[118] Béla Bollobás. *Modern graph theory*, volume 184. Springer Science & Business Media, 2013.

[119] Norman Biggs. *Algebraic graph theory.* Cambridge university press, 1993.

[120] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer Science & Business Media, 2001.

[121] Chris Johnson. Basic Research Skills in Computing Science. *Department of Computer Science, Glasgow University, UK*, 2006.

[122] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering.* Springer Science & Business Media, 2012.

[123] Robert L Glass. The software-research crisis. *IEEE Software*, 11(6):42–47, 1994.

[124] Robert A Stebbins. *Exploratory research in the social sciences*, volume 48. Sage, 2001.

[125] Gordana Crnkovic. Constructive research and info-computational knowledge generation. *Model-Based Reasoning in Science and Technology*, 314:359–380, 2010.

[126] Venkataraman Ramesh, Robert L Glass, and Iris Vessey. Research in computer science: an empirical study. *Journal of systems and software*, 70(1):165–176, 2004.

[127] Robert Davison, Maris G Martinsons, and Ned Kock. Principles of canonical action research. *Information systems journal*, 14(1):65–86, 2004.

[128] Piotr Luszczek, Jack Dongarra, and Jeremy Kepner. Design and implementation of the HPC Challenge benchmark suite. *CT Watch Quarterly*, 2(4A), 2006.

[129] OSU Micro-benchmark Suite. http://mvapich.cse.ohio-state.edu/benchmarks/, accessed July 1, 2017.

[130] T. Hoefler, T. Mehlan, A. Lumsdaine, and W. Rehm. Netgauge: A Network Performance Measurement Framework. In *Proceedings of High Performance Computing and Communications (HPCC)*, volume 4782, pages 659–671. Springer, Sep. 2007.

[131] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Russell L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The NAS parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[132] Ernst Gunnar Gran and Sven-Arne Reinemo. InfiniBand congestion control: modelling and validation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 390–397, 2011.

[133] OMNeT++ InfiniBand Flit Level Simulation Model. http://ch.mellanox.com/page/omnet, accessed July 1, 2017.

[134] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[135] András Varga. Using the OMNeT++ discrete event simulation system in education. *IEEE Transactions on Education*, 42(4):11–pp, 1999.

[136] Bartosz Bogdanski, Sven-Arne Reinemo, Frank Olaf Sem-Jacobsen, and Ernst Gunnar Gran. sFtree: A fully connected and deadlock-free switch-to-switch routing algorithm for fat-trees. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4):55, 2012.

[137] Ernst Gunnar Gran, Eitan Zahavi, Sven-Arne Reinemo, Tor Skeie, Gilad Shainer, and Olav Lysne. On the relation between congestion control, switch arbitration and fairness. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 342–351. IEEE, 2011.

[138] Jesus Camacho Villanueva, Tor Skeie, and Sven-Arne Reinemo. Routing and Fault-Tolerance Capabilities of the Fabriscale FM compared to OpenSM. 2015.

[139] Bartosz Bogdanski, Bjorn Dag Johnsen, and Sven-Arne Reinemo. Multi-Homed fat-tree routing with InfiniBand. In *22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 122–129. IEEE, 2014.

[140] Bartosz Bogdanski, Bjørn Dag Johnsen, Sven-Arne Reinemo, and Frank Olaf Sem-Jacobsen. Discovery and routing of degraded fat-trees. In *13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 697–702. IEEE, 2012.

[141] E. Zahavi. Fat-tree routing and node ordering providing contention free traffic for MPI global collectives. *Journal of Parallel and Distributed Computing*, 72(11):1423–1432, 2012.

[142] Carolyn Sher-DeCusatis, Inayath Syed, and Kirk M Anne. A Comparison Between Two Link Layer Networking Protocol Models: the OMNeT++ InfiniBand and Ethernet Models, 2010.

[143] Jens Domke, Torsten Hoefler, and Satoshi Matsuoka. Fail-in-place network design: interaction between topology, routing algorithm and failures. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 597–608. IEEE, 2014.

[144] Timo Schneider, Torsten Hoefler, and Andrew Lumsdaine. ORCS: An oblivious routing congestion simulator. *Indiana University, Computer Science Department, Tech. Rep*, 2009.

[145] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. The effect of network noise on large-scale collective communications. *Parallel Processing Letters*, 19(04):573–593, 2009.

[146] T. Hoefler, T. Schneider, and A. Lumsdaine. Multistage switches are not crossbars: Effects of static routing in high-performance networks. In *IEEE International Conference on Cluster Computing, 2008.*, pages 116–125. IEEE, 2008.

[147] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen, and Tor 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)*, pages 35–42, 2015.

[148] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen, and Tor 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)*, pages 189–198, 2015.

[149] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen, and Tor Skeie. Efficient network isolation and load balancing in multi-tenant HPC clusters. *Future Generation Computer Systems*, 72:145–162, 2017.

[150] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdanski, Bjorn Dag Johnsen, and Tor Skeie. SlimUpdate: Minimal Routing Update for Performance-Based Reconfigurations in Fat-Trees. In *1st HiPINEB Workshop, IEEE International Conference on Cluster Computing (CLUSTER)*, pages 849–856. IEEE, 2015.

[151] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen, Tor Skeie, and Evangelos Tasoulas. Compact network reconfiguration in fat-trees. *The Journal of Supercomputing*, 72(12):4438–4467, 2016.

[152] Evangelos Tasoulas, Feroz Zahid, Ernst Gunnar Gran, Kyrre Begnum, Bjørn Dag Johnsen, and Tor Skeie. Efficient Routing and Reconfiguration in Virtualized HPC Environments with vSwitch-enabled Lossless Networks. *Submitted to Concurrency and Computation: Practice & Experience*, 2017.

[153] Feroz Zahid, Amir Taherkordi, Ernst Gunnar Gran, Tor Skeie, and Bjørn Dag Johnsen. A Self-Adaptive Network for HPC Clouds: Architecture, Framework, and Implementation. *Submitted to Transactions on Parallel and Distributed Systems*, 2017.

[154] Charles Eames and Ray Eames. *A Computer Perspective: Background to the Computer Age*. Harvard University Press, 1990.

[155] James E Thornton. The CDC 6600 project. *Annals of the History of Computing*, 2(4):338–348, 1980.

[156] Jack Dongarra and Piotr Luszczek. Linpack benchmark. In *Encyclopedia of Parallel Computing*, pages 1033–1036. Springer, 2011.

[157] Maciej Besta and Torsten Hoefler. Slim Fly: A cost effective low-diameter network topology. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 348–359. IEEE, 2014.

[158] Lionel M Ni. Issues in Designing Truly Scalable Interconnection Networks. In *Proceedings of the 1996 Workshop on Challenges for Parallel Processing (ICPP)*, pages 74–83, 1996.

[159] Sharing a Common Medium: Media Access Protocols. http://web.mit.edu/6.02/www/f2010/handouts/lectures/L10-11.pdf, accessed July 1, 2017.

[160] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE transactions on Computers*, 39(6):775–785, 1990.

[161] A Bland, W Joubert, D Maxwell, N Podhorszki, J Rogers, G Shipman, and A Tharrington. Titan: 20-petaflop cray XK6 at oak ridge national laboratory. *Contemporary High Performance Computing: From Petascale Toward Exascale, CRC Computational Science Series. Taylor and Francis*, 2013.

[162] The K Computer. *Fujitsu Scientific & Technical Journal*, 48(3), 2012.

[163] Weizhen Mao and David M Nicol. On k-ary n-cubes: theory and applications. *Discrete Applied Mathematics*, 129(1):171–193, 2003.

[164] Chuan-Lin Wu and Tse-Yun Feng. On a class of multistage interconnection networks. *IEEE transactions on Computers*, 100(8):694–702, 1980.

[165] Andrzej Jajszczyk. Nonblocking, repackable, and rearrangeable Clos networks: fifty years of the theory evolution. *IEEE Communications Magazine*, 41(10):28–33, 2003.

[166] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[167] Fabrizio Petrini and Marco Vanneschi. k-ary n-trees: High performance networks for massively parallel architectures. In *Proceedings of the 11th International Parallel Processing Symposium (IPPS)*, pages 87–93. IEEE, 1997.

[168] Sabine R Öhring, Maximilian Ibel, Sajal K Das, and Mohan J Kumar. On generalized fat trees. In *Proceedings of the 9th International Parallel Processing Symposium (IPPS)*, pages 37–44. IEEE, 1995.

[169] Crispín Gómez Requena, Francisco Gilabert Villamón, María Engracia Gómez Requena, Pedro Juan López Rodríguez, and Jose Duato Marín. RUFT: Simplifying the fat-tree topology. In *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 153–160. IEEE, 2008.

[170] Eitan Zahavi, Isaac Keslassy, and Avinoam Kolodny. Quasi Fat Trees for HPC Clouds and Their Fault-Resilient Closed-Form Routing. In *Proceedings of the 22nd IEEE Annual Symposium on High-Performance Interconnects (HOTI)*, pages 41–48. IEEE, 2014.

[171] Aaron Dubrow. What Got Done in One Year at NSF's Stampede Supercomputer. *Computing in Science & Engineering*, 17(2):83–88, 2015.

[172] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. A case study of energy aware scheduling on supermuc. In *International Supercomputing Conference*, pages 394–409. Springer, 2014.

[173] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[174] Ratko V Tomic. Network Throughput Optimization via Error Correcting Codes. *arXiv preprint arXiv:1301.4177*, 2013.

[175] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 77–88. IEEE Computer Society, 2008.

[176] Unified Fabric Manager Software. http://www.mellanox.com/related-docs/prod_management_software/PB_UFM_InfiniBand.pdf, accessed July 1, 2017.

[177] B Bogdaski. *Optimized routing for fat-tree topologies*. PhD thesis, PhD thesis. University of Oslo, Norway, 2014.

[178] Gartner says worldwide public cloud services market to grow 18 percent in 2017. http://www.gartner.com/newsroom/id/3616417, accessed July 1, 2017.

[179] Robert L Grossman. The case for cloud computing. *IT professional*, 11(2):23–27, 2009.

[180] Tian Guo, Upendra Sharma, Prashant Shenoy, Timothy Wood, and Sambit Sahu. Cost-aware cloud bursting for enterprise applications. *ACM Transactions on Internet Technology (TOIT)*, 13(3):10, 2014.

[181] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L Santoni, Fernando CM Martins, Andrew V Anderson, Steven M Bennett, Alain Kagi, Felix H Leung, and Larry Smith. Intel virtualization technology. *Computer*, 38(5):48–56, 2005.

[182] Robert J. Creasy. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development*, 25(5):483–490, 1981.

[183] Aurelio Bermúdez, Rafael Casado, Francisco J Quiles, Timothy Mark Pinkston, and Jose Duato. On the infiniband subnet discovery process. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, pages 512–517. IEEE, 2003.

[184] Institute of Electrical and Electronics Engineers. 802.1 Q/D10, IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks., 1997.

[185] Peyman Oreizy, Michael M Gorlick, Richard N Taylor, Dennis Heimhigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S Rosenblum, and Alexander L Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, 14(3):54–62, 1999.

[186] M Bichier and K-J Lin. Service-oriented computing. *Computer*, 39(3):99–101, 2006.

[187] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, 4(2):14, 2009.

[188] Paul Horn. Autonomic computing: IBMś Perspective on the State of Information Technology. 2001.

[189] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[190] Autonomic Computing et al. An architectural blueprint for autonomic computing. *IBM White Paper*, 31, 2006.

[191] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger M Kienle, Marin Litoiu, Hausi A Müller, Mauro Pezzè, and Mary Shaw. Engineering Self-Adaptive Systems through Feedback Loops. *Software engineering for self-adaptive systems*, 5525:48–70, 2009.

[192] Pieter Vromant, Danny Weyns, Sam Malek, and Jesper Andersson. On interacting control loops in self-adaptive systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 202–207. ACM, 2011.

[193] Gerald Tesauro. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1), 2007.

[194] Daniel S Katz and Xiaobo Zhou. Leading-edge research in cluster, cloud, and grid computing: Best papers from the IEEE/ACM CCGrid 2015 conference, 2017.

[195] Jose Duato, Olav Lysne, Ruoming Pang, and Timothy Mark Pinkston. A theory for deadlock-free dynamic network reconfiguration. Part I. *IEEE Transactions on Parallel and Distributed Systems (IPDPS)*, 16(5):412–427, 2005.

[196] Dan Teodosiu, Joel Baxter, Kinshuk Govil, John Chapin, Mendel Rosenblum, and Mark Horowitz. Hardware fault containment in scalable shared-memory multiprocessors. *ACM SIGARCH Computer Architecture News*, 25(2):73–84, 1997.

[197] Frank Olaf Sem-Jacobsen and Olav Lysne. Topology agnostic dynamic quick reconfiguration for large-scale interconnection networks. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 228–235. IEEE Computer Society, 2012.

[198] Kim, John and Dally, William J. and Abts, Dennis. Flattened butterfly: A cost-efficient topology for high-radix networks. *SIGARCH Computer Architecture News*, 35(2):126–137, 2007.

[199] Dennis Abts, Michael R Marty, Philip M Wells, Peter Klausler, and Hong Liu. Energy proportional datacenter networks. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 338–347. ACM, 2010.

[200] Adapting InfiniBand for High Performance Cloud Computing. https://www.nextplatform.com/2017/01/17/adapting-infiniband-high-performance-cloud-computing/, accessed July 1, 2017.

[201] Jie Zhang, Xiaoyi Lu, and Dhabaleswar K Panda. High-Performance Virtual Machine Migration Framework for MPI Applications on SR-IOV Enabled InfiniBand Clusters.

In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 143–152. IEEE, 2017.

[202] Nelly Bencomo, Robert B France, Betty HC Cheng, and Uwe Aßmann. *Models@runtime: foundations, applications, and roadmaps*, volume 8378. Springer, 2014.

[203] Danny Weyns, M Usman Iftikhar, Didac Gil De La Iglesia, and Tanvir Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, pages 67–79. ACM, 2012.

[204] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. Model evolution by run-time parameter adaptation. In *Proceedings of the 31st International Conference on Software Engineering*, pages 111–121. IEEE Computer Society, 2009.

[205] Antonio Filieri, Lars Grunske, and Alberto Leva. Lightweight Adaptive Filtering for Efficient Learning and Updating of Probabilistic Models. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 200–211. IEEE Press, 2015.

[206] Timothy Mark Pinkston and Sugath Warnakulasuriya. On deadlocks in interconnection networks. In *ACM SIGARCH Computer Architecture News*, volume 25, pages 38–49. ACM, 1997.

Part II

Research Papers

List of Publications

Paper I	A Weighted Fat-Tree Routing Algorithm for Efficient Load-Balancing in InfiniBand Enterprise Clusters 23rd *Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), 2015.*
Paper II	Partition-Aware Routing to Improve Network Isolation in InfiniBand Based Multi-tenant Clusters 15th *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2015.*
Paper III	Efficient network isolation and load balancing in multi-tenant HPC clusters *Future Generation Computer Systems (FGCS), Volume 72, 2017.*
Paper IV	SlimUpdate: Minimal Routing Update for Performance-Based Reconfigurations in Fat-Trees 1st *IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), held in conjunction with IEEE International Conference on Cluster Computing (CLUSTER), 2015.*
Paper V	Compact network reconfiguration in fat-trees *The Journal of Supercomputing (JSC), Volume 72, Issue 12, 2016.*
Paper VI	Efficient Routing and Reconfiguration in Virtualized HPC Environments with vSwitch-enabled Lossless Networks Submitted to *Concurrency and Computation: Practice & Experience (CONCURRENCY), Wiley, 2017.*
Paper VII	A Self-Adaptive Network for HPC Clouds: Architecture, Framework, and Implementation Submitted to *IEEE Transactions on Parallel and Distributed Systems (TPDS), 2017.*