Towards High Performance Dynamic Cloud Environments

Evangelos Tasoulas

Doctoral Dissertation

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

June, 2017

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1922

ISSN 1501-7710

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

Abstract

The advent of the Internet of Things, sensor and social networks, to mention just a few examples, all contribute towards the solid establishment of the Big Data era. High Performance Computing (HPC) becomes necessary for the efficient processing of the massive amounts of data our society generates, and cloud computing is a critical component to deliver this processing power to a broader audience that cannot afford to acquire and maintain such complex computing systems themselves. However, HPC specific technology and performance is not yet apt to be delivered efficiently over highly flexible and dynamic environments, as typically are the virtualized cloud infrastructures.

In this thesis, we address challenges that arise in high performance dynamic cloud environments, that are equipped with HPC specific technology, in the context of networking and virtualization. We use InfiniBand, a high performance lossless interconnection network as the basis of our research, and first show that lossless networks pose prime challenges when the nature of the infrastructure is very dynamic, i.e. exhibits continuous changes. Then we propose a network I/O virtualization architecture, the InfiniBand vSwitch architecture, that can make lossless network technologies more favorable in the cloud. Moreover, we propose different network reconfiguration methods to enable performance-driven reconfigurations in very large network topologies that are commonly found in data centers. Performance-driven reconfigurations are frequently needed to adapt to unpredictable workload changes resulting from the shared and on-demand nature of a cloud platform, or when cloud providers employ live migration of virtual machines to optimize the resource usage of their infrastructure. Last but not least, we propose a new Quality-of-Service metric, called *delay*, to capture the directly observable service degradation in consolidated cloud environments. We suggest that the *delay* can be used as a direct service level agreement metric between cloud providers and cloud tenants.

Acknowledgements

First, I would like to express my gratitude to my three supervisors; Ernst Gunnar Gran, Tor Skeie and Kyrre Begnum. Without their positiveness, supervision, suggestions and encouragement, my PhD journey would not have been so fruitful and enjoyable during both the good and hard times. Then I would like to thank Bjørn Dag Johnsen from Oracle Norway, our closest collaborate in the ERAC project, the project that mainly funded this thesis, for his enthusiastic attitude and participation in the important discussions that shaped the direction of my work and introduced industrial relevance. My appreciation goes as well to Feroz Zahid for being the most easygoing colleague whom I could imagine sharing office with, and a brilliant and ambitious associate, and Sven-Arne Reinemo for his supervision during my early days.

A special thank you must be directed to Hårek Haugerud, Anis Yazidi, Hugo Lewi Hammer, Laurence Marie Anna Habib and the whole NETSYS group at the Oslo and Akershus University College for being so supportive towards me since I started my Master program and all the way through my PhD, even if they were not directly involved in the ERAC project. It must be noted that without their partial funding it would have been much harder for me to reach the finishing line.

Last but not least, I want to communicate my utmost respect, love, and appreciation to my parents, Alexandros and Anna Tasoula, for their infinite support and the unconditional love they always gave to me. To the rest of my family and my little niece, Anna-Maria, for making me laugh so easily; To Dr. Xinoula Song for believing in me, supporting me in so many different ways and praising me unconditionally; To my friend and soon-to-be Dr. Dimitrios Agiakatsikas (also known as Lakis) for his eternal wisdom and discussions we had in so many different topics of sheer importance during our PhD moments; To all of the anonymous reviewers of this thesis. I would like to conclude by thanking my very good late friend and the most kind hearted person I met so far, Karolos Trivizas. During the short eight years that I knew him, his love towards me and his experienced advices made him feel like a second father to me. I feel grateful to have met you, and your memory will be with me always.

Table of Contents

Appended Papers

Chapter 1

Introduction

This thesis aims to address challenges that arise in high performance cloud computing environments in different layers of a cloud stack. A core component for delivering high performance in the cloud is the interconnection network. High performance lossless networks typically provide lower latency and higher bandwidth when compared to lossy networks. When cloud environments are combined with lossless interconnection network technologies, like InfiniBand (IB) [1], scalability issues with respect to managing the network come into sight. Lossless networks have been traditionally used in static environments, such as High Performance Computing (HPC) clusters, and the dynamic nature of the cloud challenges such network fabrics.

In this work, we first undertake network oriented challenges in the context of virtualization[1]. Then, we move further up in the cloud stack and study an evidently perceptible Quality of Service (QoS) metric, the *delay* of workloads, for cloud consolidation.

1.1 Motivation

New, tiny devices with increased computing power and embedded sensors are being deployed ubiquitously every day. Sensor networks and the Internet of Things (IoT) are more palpable than ever before and many *smart* connected products such as phones, fridges, lamps, watches and other wearables, just to name a few, have already hit the market. The number of connected devices is projected to expand from the roughly 20 billion that exist today, to 30 billion by 2020 and 80 billion by 2025, as presented by a recent International Data Corporation (IDC) report [3]. These devices as well as the rise of social media like Facebook[2] and YouTube[3] produce enormous amount of data, contributing towards the *Big Data* era we are going through.

[1]Virtualization is one of the core technologies powering up Infrastructure as a Service (IaaS) [2] clouds.
[2]Facebook generated four new Petabytes of data per day as of 2014 [4] and had 1.86 billion active users by the end of 2016 [5].
[3]YouTube users upload more than 400 hours of video per minute [6].

Big Data is a term commonly used to describe the polynomial growth of data generation we have seen in recent years. In 2005 the amount of data created and copied was 130 Exabytes [7]. In 2015 that number had grown to approximately 10 Zettabytes, and it is estimated that by 2025 we will be producing 180 Zettabytes (180 trillion Gigabytes) annually [3]. Notwithstanding the great amounts of data we produce, our ability to analyze those data is poor. Only half percent (0.5%) of the generated data is ever analyzed [7].

The three main Big Data characteristics are widely known as the *3V's*; *Volume*, *Variety* and *Velocity* [8]. The 3V's bring several research challenges along the way loosely summarized as: how to process the huge *V*olume (Exabytes, Zettabytes) of *V*arying data (video, text, structured, unstructured) with high *V*elocity (real time generating and processing interval requirements) efficiently? How to turn more than half percent of the generated data into value by extracting accurately what is needed in a timely manner?

It becomes noticeable that performance is critical to address the Big Data challenges, and for this reason HPC clusters are typically used for efficient Big Data analytics [9]. On the other hand, HPC clusters have a very high deployment and maintenance cost that individuals or small organizations cannot afford. However, with the emergence of cloud computing, the Computer Science (CS) society tends to agree that there will be a convergence of HPC, Big Data and the Cloud, with the Cloud performing as the vehicle for delivering the associated services to a broader audience [10, 11].

Cloud computing, or simply cloud, is a paradigm shift in the Information Technology (IT) sector. Cloud computing refers to the usage of computing resources delivered as services over a network in a server-centric model. The infrastructure is usually[4] not owned or maintained by the cloud clients[5], but rented on demand. The cloud brings some attractive features such as: a) resource elasticity and consolidation that provide both monetary benefits for the cloud tenants and environmentally friendlier (greener) computing, b) infrastructure management as a service that can be controlled and automated with software, c) the opportunity for cloud users to cost-effectively try out new ideas that would otherwise require a large upfront investment in hardware[6], and d) lifting the infrastructure maintenance burden from the end user since the cloud provider is taking care of the infrastructure. Due to the valuable advantages the cloud can offer, it is no coincidence that Cisco predicts the total global data center traffic to reach 15.3 Zettabytes annually by 2020, and 92% of all workloads to be processed in the cloud by then [15].

Server or hardware virtualization[7] is arguably one of the core cloud components [2]. Virtualization breaks the one-to-one relationship between the Operating System (OS)

[4]Unless the cloud is private or hybrid [12].

[5]The cloud clients can also be referred to as cloud *tenants*.

[6]Dropbox Inc. is a shinny example of a company that grew on the shoulders of a big cloud provider. Dropbox started small without investing in its own hardware, as that would require a major upfront investment to provide a reliable, robust and highly durable service for storing files. Since its inception, Dropbox was storing files in the Amazon Web Services (AWS) [13] cloud platform and only when Dropbox became big enough did Dropbox invest on its own data centers [14].

[7]There are several types of virtualization. Some examples include the *network virtualization* [16], *service virtualization* [17] and *server/hardware virtualization* [18]. When we plainly refer to *virtualization* in the rest of this work we are referring to *server/hardware virtualization*.

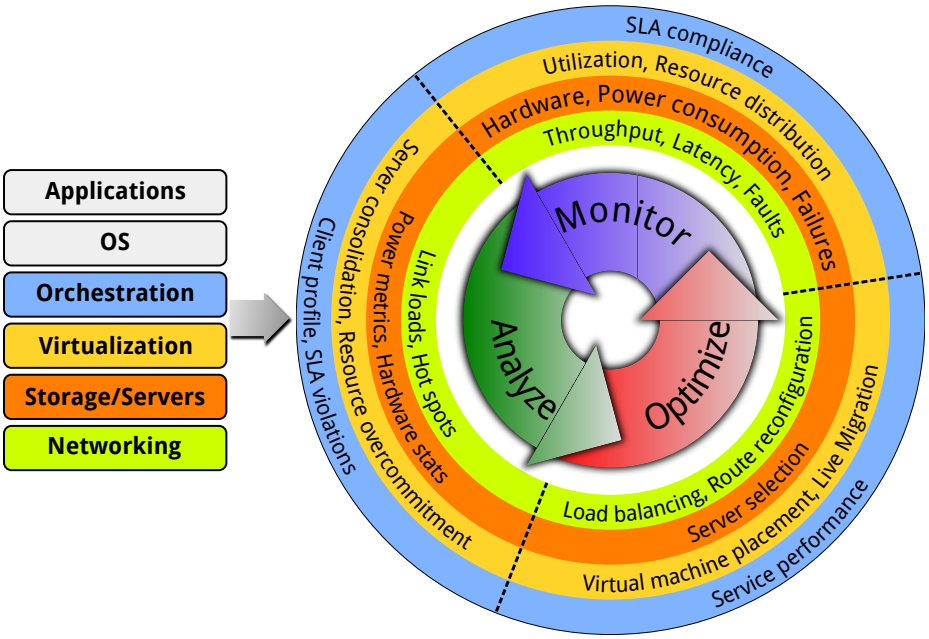


Fig. 1.1: Different layers and research objectives of interest for the corresponding layer of a self-adaptive cloud.

and the hardware by means of logical abstraction of the hardware that is exposed to the virtualized Operating Systems (OSs) in the form of Virtual Machines (VMs) [18]. As a result more OSs can run simultaneously, but at the same time independently in isolated VMs, on top of the same physical hardware. At a glance the outcome of the hardware abstraction is increased server consolidation, but more importantly, the computing resources can be treated as virtual entities that remove the hard physical boundaries of the physical systems. The removal of the physical boundaries opens up a whole new world of ways to handle computing infrastructure since the hardware can be treated similarly to software. Most of the cloud benefits that were mentioned in the previous paragraph can be directly attributed to virtualization. Further details about the cloud concepts and virtualization features will be given in Chapter 2.

Along with the benefits of the cloud and virtualization, there also come challenges. The several layers of abstraction in consideration of achieving a fully self-adaptive software defined architecture, as shown in Fig. 1.1, are desired in order to allow for dynamic optimization of resources [19, 20], but add to the complexity of the cloud components, opening up new frontiers in several research areas. Moreover, the added overhead due to the additional layer of virtualization between the OS and the hardware has a performance impact. As a consequence, the performance of a fully virtualized cloud is not on par with that of physical, non-virtualized systems. The performance gaps are even larger in the HPC domain where HPC specific technologies such as lossless interconnection networks are common, but not yet ready to be fully virtualized without sacrificing cloud flexibility [21]. Correspondingly, cloud providers have not adopted true HPC clouds yet that can perform on par with traditional HPC systems [22, 23], a necessary move in order to accommodate high performance workloads efficiently in the cloud, and therefore bring HPC and efficient Big Data analytics access to an extended audience.

Cloud Layers

A fully virtualized environment with several layers of abstraction is needed to accomplish a software defined architecture that uncouples services from location. This uncoupling is necessary for the realization of self-adaptive clouds that are able to monitor their state, analyze, and eventually optimize the offered services by rearranging/reallocating resources. Fig. 1.1 liberally presents the layers of a typical cloud infrastructure with the rectangles on the left of the figure, and corresponding research objectives for each layer[8] in the context of the monitor-analyze-optimize loop[9] on the right side of the figure. Starting from the bottom layer and working up we meet the *Networking* layer. The network is an integral part of any cloud infrastructure, as by definition all the cloud resources are served to the clients through the network. However, this work focuses in intra-cloud challenges related to the virtualized network resources that are allocated to cloud tenants, thus, with the term *Networking* we refer to that aspect of the interconnection network itself, and not the networking resources that are used for the management of the cloud data center. One layer above the network we have the *Storage/Servers* layer. This is the layer where the physical hardware that delivers the computing resources to the cloud users is located. On the third layer we have the layer of *Virtualization* that is responsible for the decoupling of the services from the underlying physical infrastructure, i.e. servers and storage etc. Then comes the *Orchestration* layer where the operational intelligence is implemented. The responsibility of the orchestration layer is to enforce security and usage policies for the cloud tenants, ensure Service Level Agreement (SLA) compliance between the tenants and the cloud provider, tracking of resource utilization and billing. Above the orchestration layer sit the *OS* and *Applications* that are being deployed on demand to the cloud by the tenants via the orchestration layer.

Each of these layers have different objectives in the monitor-analyze-optimize loop. For example, in the monitoring phase the orchestration layer is responsible to monitor the SLA compliance and pass the necessary metrics to the next phase for analysis. In the Analysis phase the client profiles should be checked and potential SLA violations must be identified. The optimization phase should try to reallocate resources if needed based on the input of the analysis with the ultimate goal to improve the service performance. Note that in most situations coordination is needed between different layers in order to improve overall performance.

Plenty of research has been done in the different objectives for each of the cloud layers in the effort to materialize self-adaptive cloud infrastructures [26, 27, 28], but not in the context of clouds with HPC specific technologies such as lossless HPC interconnection networks. To be more distinct, modern virtualized clouds that offer high performance based

[8]Note that the different cloud layers and corresponding research objectives in Fig. 1.1 are presented as a high-level overview of cloud architectures. We do not address issues in all of the cloud layers in this thesis. Details about the focus of the thesis are to be found in Section 1.2.

[9]For the sake of simplicity, the monitor-analyze-optimize loop that is used in Fig. 1.1 is a relaxed presentation of the widely recognized Monitor-Analyze-Plan-Execute (MAPE) [24] classification for autonomic computing, where the *Plan/Execute* phases are presented as the *Optimize* phase. The MAPE classification is common in self-adaptive cloud environments [25].

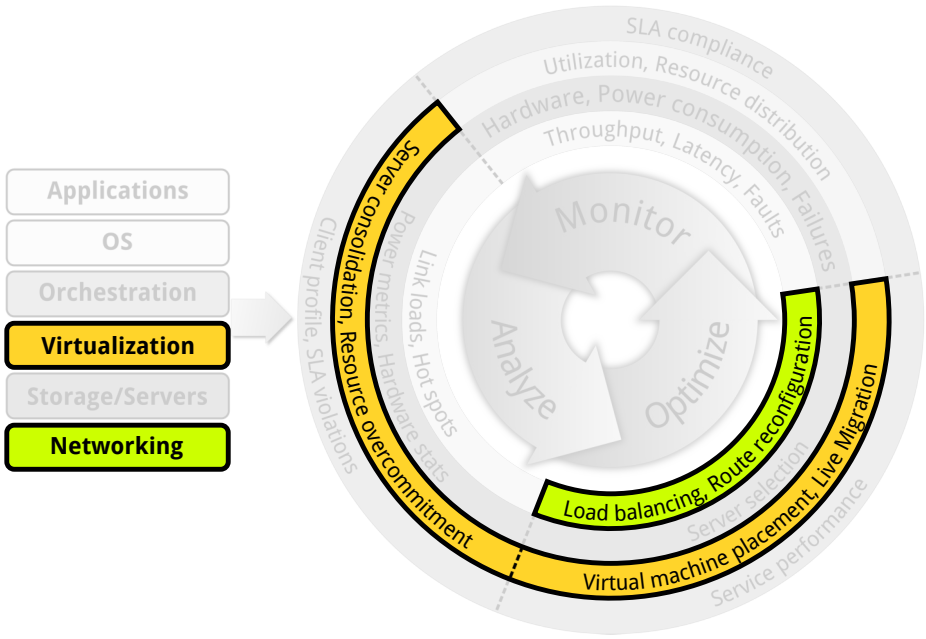


Fig. 1.2: Highlighting research objectives that this thesis is focusing on. We target the networking and virtualization layers from the perspective of a lossless network when virtual machines are migrating in the optimization phase, as well as the virtualization layer with respect to resource overcommitment in the analysis phase.

on lossless networks have been demonstrated in scientific papers [29, 30, 31, 32], but not much effort has been put in the dynamic nature and self-adaptation of a cloud when a cloud is based on such network technologies as explained in more detail in Section 1.2.

1.2 Research Challenges Addressed in this Thesis

This thesis targets a subset of the research objectives of the monitor-analyze-optimize loop in self-adaptive clouds. As highlighted in Fig. 1.2 we concentrate in the networking and virtualization (from the perspective of the network) layers of the optimization phase, as well as the virtualization layer with respect to resource overcommitment in the analysis phase. In particular, we first address challenges related to high performance lossless interconnection network technologies in the context of dynamic virtualized cloud environments. Once we have demonstrated techniques that would allow for efficient live migration of VMs that use a lossless network, as well as network performance optimization techniques that are needed due to the constant workload or infrastructure changes in a cloud environment, then we look at the problem of consolidation from the orchestration perspective. We use IB as the network interconnect of choice[10] for demonstrating our prototype implementations, but the concepts and challenges presented in this thesis are related to the nature of lossless high performance interconnection networks, thus, even similar competing technologies could benefit from our results [34, 35]. As long as something is purely IB specific it will be clearly

[10]IB is a popular lossless interconnection network technology holding a significant 35.4% market share in the list of the top 500 supercomputers [33] as of June 2017.

mentioned throughout the text in this document.

Although the operating principles of lossless networks in general improve performance, the fact that packets are stored in a sender node and are neither dropped nor forwarded until there is available buffer space in the receiver node, introduces the potential for deadlock situations if routing cycles (loops) exist[11] [36]. Ultimately, the network may come to a halt until the deadlock is resolved. To prevent deadlocks from happening lossless networks typically engage a management authority, also known as Subnet Manager (SM). The SM is responsible for the overall arbitration of the network: discover, configure the nodes with layer-two Local Identifier (LID)[12] addresses, calculate deadlock-free routing paths based on the LID assigned to each node, provide path resolutions and once the network is running, continuously monitor the health of the network.

In large subnets the SM can become a bottleneck that prevents scalable management of the subnet. For two peers to communicate, they need to know key characteristics of the communication path, such as which is the maximum supported Maximum Transmission Unit (MTU) or speed rate. A path resolution performed by the SM provides this information to the peers. As we demonstrate in Paper I[13] one scalability issue can emerge from the path resolutions that increase polynomially as the size of a subnet increases. Cloud environments are very dynamic with constant workload changes and need for re-optimizations when VMs live migrate. Especially for the case where a Virtual Machine (VM) live migrates, the LID address of the VM will change as the LID is shared with the physical host in current generation Host Channel Adapters (HCAs)[14] [38, 39]. This layer-two address change will force the peers communicating with the migrated VM to ask for the new path characteristics to reconnect, a behavior that leads us to the first Research Question (RQ) of this thesis:

> *RQ 1: In dynamic cloud environments VMs can live migrate to optimize resource usage and performance. With the readily available technology in the IB architecture the VMs do not get a dedicated LID address, i.e. when a VM migrates, its LID address will change. What are the implications on the subnet management scalability?*

RQ 1 is addressed in Paper I where we show that the SM performance can get a serious hit when VMs migrate and their LID addresses change. We show that it is possible to remediate the scalability issues if we introduce a client-side cache for paths that have already been resolved. A necessary condition for the cache to work is that the VMs should have dedicated LID addresses. But then again, with dedicated LID addresses in the VMs the SM would have to reconfigure the network and reroute the traffic after each live migration. Reconfiguring the network while running –an operation known as *dynamic reconfiguration*– is a costly and challenging task. When a lossless network is reconfigured,

[11] The necessary background about the operating principles of lossless networks is located in section 2.3.

[12] The terminology that is used throughout this thesis (SM, LID, etc.) is based on IB. However, most of the terms are similar or identical even in competing technologies that are based on the same lossless principles [1, 37].

[13] The six research papers written as part of the PhD work leading to this thesis are all attached in the last chapter of this thesis.

[14] A Host Channel Adapter (HCA) is the term the IB specification uses to refer to a network card.

deadlock-free routes have to be recalculated and distributed to all switches. This task can take several minutes depending on the network size and topology [40, 41]. Normally, HPC clusters are static environments and once deployed the SM rarely intervenes to reconfigure the network unless critical faults are detected. In a cloud environment reconfigurations are common. Therefore, the next two research questions are:

> *RQ 2: What would the implications be of having a network architecture that would allow VMs to be assigned with their own layer-two addresses and what should one take into account when designing such an architecture?*
>
> *RQ 3: If an architecture that allows VMs to be assigned with their own layer-two addresses exist, a reconfiguration of the network would be needed after each VM live migration to redirect the traffic to the new destination of the VM. How would it be feasible to have continuous cloud optimizations with VM live migrations in large subnets if one takes into account that each reconfiguration may impose a large overhead in the network and take minutes to complete?*

In Paper II we present a vSwitch network Input/Output Virtualization (IOV) architecture that takes into account different scalability aspects of the network. Firstly, the vSwitch architecture allows VMs to get a dedicated layer-two address, thus, whenever a migration of a VM happens the scalability issues that have been presented in Paper I can be resolved. The first part of Paper II provides answers to RQ 2. Then, in the same paper, we present analytically a topology-agnostic reconfiguration technique that would allow for very quick reconfigurations even in very large subnets. In Paper V we complement our proposed vSwitch architecture by providing an even more efficient topology-aware reconfiguration technique, and we discuss routing considerations for vSwitch-based subnets. The latter part of Paper II as well as Paper V provide insights for RQ 3.

When VMs migrate or cloud tenants spawn new VMs the traffic patterns of the network change. This diverse and continuously changing workloads are the norm and not the exception in cloud environments. In several situations the performance can be improved with a dynamic network reconfiguration. Nonetheless, reconfiguring a large subnet is burdensome. As previously mentioned a reconfiguration can take several minutes depending on the network size and topology. The research question for this challenge is:

> *RQ 4: Reconfiguration of the network can take several minutes in large subnets. Due to the dynamic nature of cloud environments performance-driven reconfigurations are more relevant than ever before. How could one reduce the reconfiguration overhead to allow up to several performance-driven reconfigurations per minute if needed?*

One solution for RQ 4 is presented in Paper III. A topology-aware method that allows for fast reconfiguration of the network in order to keep up with constant changes is demonstrated. The method identifies the switches and nodes that are located in a sub-part of the network that could benefit from a reconfiguration, and reconfigures only that sub-part by orders of magnitude faster when compared to a full subnet reconfiguration. Moreover, different routing algorithms can be used as fit in different sub-parts of the network. In Paper IV another solution is presented where a meta-database with alternative paths –that

can be used during a reconfiguration– is generated during the initial configuration phase. When a reconfiguration is needed the network can be reconfigured quicker by selecting paths from the alternative pre-calculated options available in the meta-database.

Up until now we added building blocks in different levels of the interconnection network layer (architectural level, network management) for dynamic environments. Now we move further up in the cloud hierarchy and look at the problem of performance fluctuation in the cloud which is mostly a result of running unpredictable workloads and overcommitting resources [42, 43]. It is known that HPC workloads can have tight deadlines [44, 43], and after all, what the user typically experiences when the performance fluctuates is the varying delay of the workload execution. This problem leads us to the last research question for this thesis:

> *RQ 5: Cloud workloads can be time constrained. A metric that users directly perceive when executing a well known workload, is how much time did the computation need to complete and if the execution has been delayed. How can cloud providers increase consolidation in cloud environments without compromising the performance expected by the customers?*

In Paper VI we introduce the concept of *delay* as a QoS metric in our quest to address RQ 5. The delay of a workload is an evidently perceptible metric from both the cloud provider and the cloud tenants, thus, the delay could serve as a fair SLA metric for consolidated cloud environments. This work provides preliminary insight about how the delay of workloads could be formulated, and demonstrates a simple bin packing method for consolidation of VMs based on the *delay*.

For a quick reference, Fig. 1.3 maps the different RQs and Papers to the corresponding layers of a self-adaptive cloud.

1.3 Research Methods

Generally, there are two methodological approaches in Computer Science: *Theoretical* and *Experimental* [45, 46, 47, 48, 49, 50]. In theoretical CS, conceptual and formal modeling, and mathematical proofs of propositions are commonly used. In experimental CS the research is mainly comparative and is based on methods such as benchmarking, prototyping, simulating, and analyzing the results. According to need, this thesis uses several research methods that mostly belong to the experimental CS class. In particular, *Lab-based Experiments*, *Concept Implementation* or *Prototyping*, *Simulation*, *Data Analysis* and *Conceptual Analysis/Mathematical (CA/M)* have been used.

The purpose of the *Lab-based Experiments* can be twofold. First, lab-based experiments are controlled and can be carefully designed to collect data and study the real-life behavior of a particular element of interest in complex systems. Second, lab-based experiments can be used to compare the performance of a newly-proposed system with existing systems. For example, in Paper I we designed and used a lab-based experiment to benchmark and

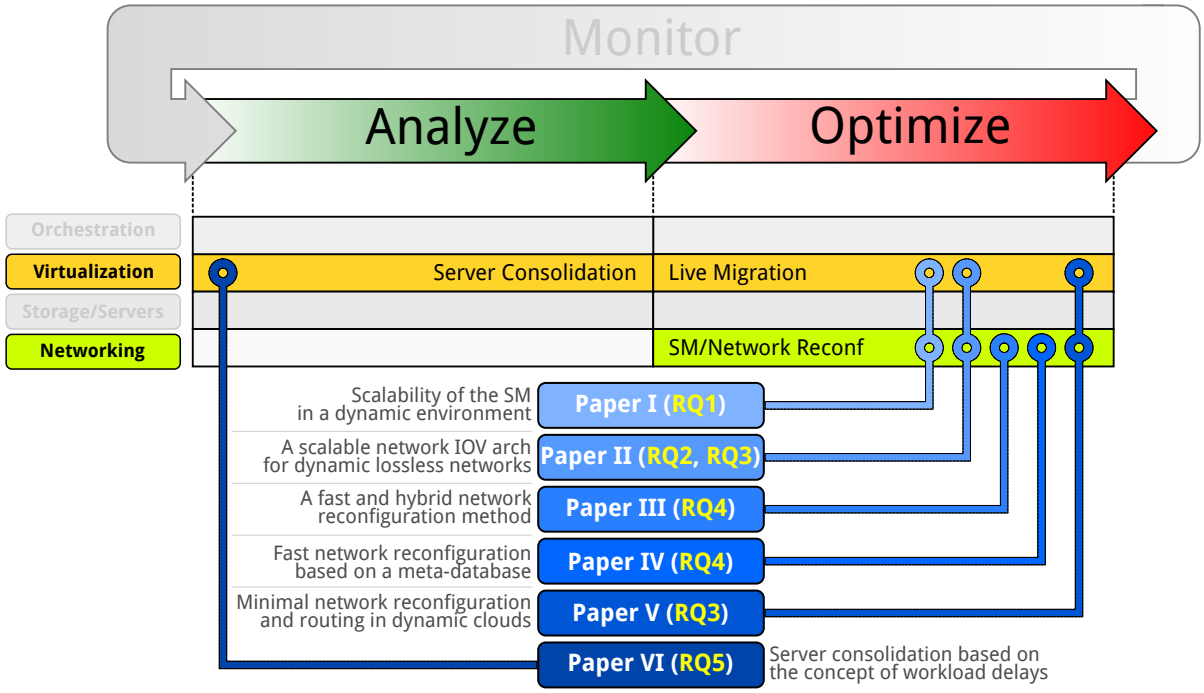


Fig. 1.3: A flattened version of Fig. 1.2, that maps the different RQs and Papers to the corresponding layers of a self-adaptive cloud.

study the scalability and behavior of the IB SM for the particular case where too many path resolution queries need to be served. In the same work we set up experiments to compare our proposed prototype solution, that introduces a client-side path resolution cache to reduce the load imposed to the SM, with the behavior of the unmodified system. Experimentation is a valuable research method that is commonly used to verify or refute the claims of a hypothesis [49]. Lab-based experiments have been thoroughly used in all of the papers in this thesis.

Concept Implementation or *Prototyping* is used to provide proof of concepts, and this research method has been used in all of the proposed concepts in this thesis. Most theoretical approaches involve some level of modeling abstraction. Modern computing systems are very complex and it is not possible to model every aspect of each studied system in detail. As such, something that is proved theoretically feasible may not be practical to implement due to unexpected behavior of system parts that have been left out in the abstract model. A prototype can reduce the gap between theory and what is actually possible in a real-life scenario. Prototyping can also reveal potential mis-modeling in the theoretical part. Depending on the level of modeling abstraction of a research idea, a prototype can be hard to implement. If a radical concept needs several other pieces that do not yet exist, a concept implementation may not be feasible at all. The operational level of the prototypes presented in this thesis differs with the nature of each research idea. For example, in Paper II a non-existing hardware architecture is proposed. The implemented prototype that accompanies this work tests the practicality and performance of the proposed reconfiguration method for the proposed architecture by emulating the proposed IB vSwitches with real switches, and the VMs that would be connected to the

vSwitches with real nodes connected to the switches. One the other hand, in Paper III the presented prototype is a more straight-forward implementation since the proposed solution can be applied and fully demonstrated with readily available hardware. It is noteworthy that although prototyping is a valuable research method, only a demonstration of a prototype cannot stand alone as a scientific research method when not accompanied by detailed experiments and analysis of the results [50].

Simulations can reproduce the behavior of a system in study by using a model. Accurate modeling of complex real-life systems is almost impossible, thus, a simulation model is usually abstract and captures only the variables of the system that are required for the study. Therefore, the results of a simulation cannot offer a real-life certainty, but depending on how accurately the system variables have been captured and modeled, the results can be very close to reality. Although simulations require lots of computing resources and can be very slow, the insights provided by a simulation may be impossible or cost ineffective to acquire otherwise. Simulations are typically used when one wants to study system variables that cannot be easily observed by other means (e.g. the necessary tools to observe or measure a phenomenon do not yet exist), for feasibility studies when the system one wants to look at does not exist and wants to find out if it even makes sense to try to build it (e.g. before building a prototype aircraft with a radical new design), or when concepts that would otherwise require big upfront investments need to be studied (e.g. ideas need to be evaluated in an expensive system that one cannot get hold of). This thesis uses simulations in the latter context, i.e. evaluating how our prototype concepts would perform in large-scale systems that we cannot afford to get access to[15]. The Oblivious Routing Congestion Simulator (ORCS) [51] has been used for testing the routing quality of our proposed routing/reconfiguration algorithms in Paper III, Paper IV and Paper V. As part of Paper III we made significant contributions back to the ORCS project. The Infiniband Fabric Simulator (ibsim)[16] has also been used to test the algorithm execution performance of different routing/reconfiguration methods in Paper II, Paper III and Paper V. To fulfill the evaluation requirements for Paper V, we also created a new simulator, the *vSwitchMigrationSim* simulator, that was used to simulate live migration of VMs and reconfigure the network as our proposed algorithms would do in a network powered by our proposed vSwitch architecture.

The goal of *Data analysis* is to inspect a series of collected data and try to extract useful information that will provide new insight. During the analysis phase different statistical tools, such as standard deviation, mean, probability distribution fitting, quantiles of the data etc., are commonly used to identify patterns. In the *evaluation* sections of all of the papers in this work we analyzed the data from our results, that are based on simulations and prototypes. Particularly in Paper VI, we had to deal with and analyze a very large

[15]Such large-scale systems are in general very expensive and precious production-systems where it is extremely hard to get access to experimenting with the infrastructure/system itself.

[16]Ibsim is distributed as part of the OpenFabrics Enterprise Distribution (OFED™) [52] software. Strictly speaking, ibsim is not a simulator but an *emulator*. In contrary to simulation where all parts of the simulated system are modeled, in emulation some functional part of the model is carried out by a part of the real system [53]. Thus, an emulation can provide results that reflect reality with greater certainty. In the ibsim case, the part of the real system that participates in the emulation is the SM.

	Lab-based Experiments	Prototyping	Simulation	Data Analysis	Conceptual Analysis /Mathematical
Paper I	✓	✓		✓	
Paper II	✓	✓	✓	✓	✓
Paper III	✓	✓	✓	✓	
Paper IV	✓	✓	✓	✓	
Paper V	✓	✓	✓	✓	
Paper VI	✓	✓		✓	✓

Table 1.1: Summary of which research methods have been used in each research paper.

dataset of real-life performance data (Central Processing Unit (CPU)/Memory utilization) collected from more than 1500 Linux-based systems. After the dataset was cleaned up from erroneous values, we applied our proposed concept algorithms, that are presented in the same paper, on the real-life dataset in order to base our final results on a genuine input.

Conceptual Analysis/Mathematical is a method that falls under the theoretical CS. In Conceptual Analysis (CA) one tries to break a problem into smaller well understood components. Then the necessary and sufficient conditions for the relation between the components are defined [54]. The CA/M is defined by Ramesh et al. [46] as an extension of the CA research method where mathematics are used to show the connection of the components. We have used the CA/M method to formally describe the overhead of our proposed vSwitch architecture in Paper II. Moreover, the same method was used to define the concept of *delay* that we proposed in Paper VI.

Table 1.1 provides a short summary about which research methods have been used in each research paper.

1.4 Thesis Outline

This thesis is divided into two parts. The first part contains the *Introduction* (this chapter), *Background*, *Summary of Research Papers* and *Closing Remarks* in chapters 1, 2, 3 and 4, respectively. In the introduction the topic is motivated before we unfold the research objectives, research questions and research methods that have been used. In the background chapter we provide insight to related technologies and concepts that concern this work. In the research paper summary chapter, an extended abstract for each of the research papers is presented before we provide the closing remarks where the future research directions are included. The second part contains a collection of the six research papers that have been published or submitted as part of this doctoral dissertation. These six papers reflect the actual research that has been conducted and contain details about the devised algorithms, designs, prototypes and evaluation of our results. Note that the six papers that form the second part of this thesis are the same ones that have been summarized in Chapter 3, and referred to with the "Paper I – VI" notation throughout this thesis.

Chapter 2

Background

In this chapter we introduce background technology, prior work and tools related or used in this thesis. First we discuss virtualization, as virtualization is one of the main technologies powering up cloud infrastructures. We focus in the overhead of virtualization, Input/Output Virtualization (IOV), and the feature of *live migration* of VMs that enables much of the dynamicity in the clouds. Then we discuss cloud environments, before we move into the operating principles of lossless interconnection networks and IB, where we put everything together. We conclude this chapter with a description of each one of the simulators that have been used in this thesis.

2.1 Virtualization

The term *virtualization* in computing refers to emulation of hardware, software, or services to allow abstraction and isolation from the underlying hardware, operating systems, or lower level functionalities. Server or Hardware virtualization is arguably one of the best known applications of virtualization technology and usually when the term *virtualization* is used directly, people mostly refer to hardware virtualization.

The roots of hardware virtualization and VMs can be traced back to the 60s, when IBM introduced the concept for their mainframe systems [55, 56]. The goal for IBM was to offer the means for isolated hardware sharing between different clients and software portability between different IBM systems. However, it was not before the end of the 90s, that hardware virtualization gained mainstream adoption and evolved to become one of the primary technologies that today powers modern data centers and cloud infrastructures. In particular, VMware introduced software that provided virtualization capabilities to commodity hardware based on the widely adopted x86 architecture [57].

Hardware virtualization breaks the one-to-one relationship between the OS and the hardware by means of logical abstraction of the hardware that is exposed to the virtualized OSs in the form of VMs [18]. As a result more OSs can run simultaneously, but at the same time independently in isolated VMs, on top of the same physical hardware. The software

component which is responsible to control and allocate the hardware resources needed to create and run VMs is called *Virtual Machine Monitor (VMM)* or *hypervisor* [58], while each VM can also be called a *guest*.

One of the often highlighted benefits provided by virtualization is the increased server consolidation. Server consolidation promotes greener and cheaper computing as multiple VMs can share the same physical hardware. That is, less, but more efficiently utilized servers (servers that have less idling time) can be used to serve the same workload. Server underutilization is a well known problem [59]. Yet, the new tools and features provided by virtualization, that enable novel ways for infrastructure management, are of equal importance. One such feature is the ability of VMs to live migrate to different physical locations (servers) with service downtimes in the order of milliseconds [60].

2.1.1 Live Migration

Migration of VMs to different physical hosts is one of the most prominent features of virtualization. Live or hot migration offers the ability to migrate while the VM is operational and enables a new range of infrastructure management possibilities. Live migrations are typically used for workload optimization and server consolidation [61], maintenance [60] and disaster recovery [62]. Live migration is a key feature to enable resource efficient, dynamically reconfigurable data centers [63, 64]. There are two well known live migration methods: *Pre-copy* and *Post-copy* [65].

In the pre-copy method the source hypervisor starts with the *warm-up* phase by sending memory pages of the VM-in-migration to the destination hypervisor. The VM is still in operation during this phase so some of the transferred memory pages will become dirty (change) during this process. The warm-up phase is iterative, i.e. tries to resend memory pages that have been dirtied, and typically lasts until some condition is met. For instance, the remaining dirty memory pages will not take more than x seconds to be transferred to the destination hypervisor based on the estimated transferring speed from previous memory page copying iterations. When the condition is met, the *stop-and-copy* phase is following where the VM is suspended in the source hypervisor, the remaining dirty memory pages are being transferred, and the operation is resumed at the destination. During the stop-and-copy phase the VM experiences downtime. An issue of the pre-copy scheme is that if during the warm-up phase the memory pages become dirty with a rate that is faster than the network speed, it may not be possible to meet the required condition to transition to the stop-and-copy phase.

The post-copy method follows a slightly different approach to overcome the issue of the pre-copy method, at the cost of imposing a performance penalty to the VM during the migration. The VM is transferred immediately and resumed to the destination hypervisor[1] and the memory page copying is following. When the VM requests to access a memory page that hasn't been transferred to the destination yet, a (remote) page fault fetches the page from the source.

[1] In practice there may still exist a warm-up phase but it is not necessary.

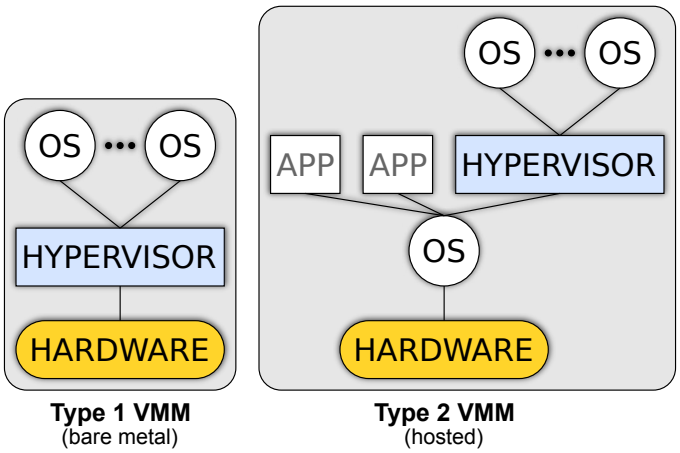


Fig. 2.1: Types of hardware virtualization hypervisors

2.1.2 Virtualization Overhead

There exist two types of hypervisors as shown in Fig. 2.1. The *Type 1* hypervisors, also known as *bare metal*, are dedicated hypervisors that sit directly on top of the hardware and the sole purpose of a system that runs a type 1 hypervisor is to deploy VMs. The *Type 2* hypervisors on the other hand run as an application on top of an existing OS. No matter which of the two types of hypervisor one is using, virtualization adds performance overhead as the OS and applications that run inside a VM access the hardware through an additional software layer, the hypervisor [66, 67].

Over the years, the situation has improved considerably as CPU overhead has been practically removed through hardware virtualization support [68, 69]; storage overhead reduced by the use of fast Storage Area Network (SAN) storages or distributed networked file systems; and network I/O overhead reduced by the use of device passthrough techniques like Single Root I/O Virtualization (SR-IOV) [70]. Although these different techniques improve performance by reducing the virtualization overhead, not all of them come without drawbacks as we will see in the next section for the particular case of network IOV.

2.1.3 Input/Output Virtualization

The purpose of IOV is to provide multiple VMs with shared and protected access to I/O resources. Typically, IOV decouples the logical device which is exposed to a VM from its physical implementation [58, 71] and there are two common approaches:

a) **Software emulation**, which is a decoupled front-end/back-end software architecture. The front-end software is a device driver placed in the VM that is able to communicate with the back-end implemented by the hypervisor.

b) **Direct device assignment**, which involves either decoupling at the hardware level or coupling of a device and no sharing between multiple VMs.

Both of the approaches have advantages and disadvantages. A brief discussion of the four

most commonly used network IOV techniques and their characteristics follows:

- **Emulated real devices** fall under the software emulation approach. Emulating real devices (such as an existing Intel network interface card for example) is the dominating technique when it comes to guest OS support, sharing ratio and supported virtualization features. VMs can run by using unmodified drivers of the emulated device, and the guest OS behaves as if it was running on real hardware. Live migration is possible with minimum network downtime in the order of milliseconds. Note that since the hardware that the VM *sees* is virtual, the VM hardware state is stored in memory as well in the software emulated techniques. Thus, live migration can migrate a VM efficiently without the need for checkpointing individual software/hardware components that are running in the VM. However, emulating real devices suffer from poor performance and adds overhead to the hypervisor since CPU cycles have to be dedicated for emulating some non existing hardware.
- **Paravirtualization** is a different software emulation approach. Paravirtualization improves performance compared to the emulated devices by exposing some optimized virtual hardware to the guest OS, rather than emulating a real device. Sharing ratio and supported virtualization features are on par –or even better due to the reduced overhead– when compared to emulated devices. The disadvantage is that special drivers need to be installed in the guest OS for the virtual hardware to be enabled.
- **Device pass-through** is a direct device assignment method that provides near to native non-virtualized performance and minimum overhead. A Peripheral Component Interconnect (PCI) device is directly attached to the VM and bypasses the hypervisor. The VMs can run by using unmodified drivers. The downside of this approach is the limited scalability as there is no sharing. One physical network card is coupled with one VM. Furthermore, currently there is no simple way to live migrate VMs without having a long network downtime in the order of several seconds.
- **Single-Root IOV (SR-IOV)** is a form of device pass-through that introduces the notion of a PCI-Express Physical Function (PF) and multiple Virtual Functions (VFs). The VFs are light-weight instances that are identical to the PF, but are not allowed to reconfigure the PCI device. The VFs are assigned to VMs. As a direct device assignment method, SR-IOV is supported by non-modified drivers and provides near to native performance while solving the problem of scalability of the full device pass-through method. The cons of direct device assignment regarding live-migration are unfortunately inherited as well [72].

2.2 Cloud Computing

Cloud computing is a term that is used ubiquitously to refer to the usage of computing resources delivered as services over a network in a server-centric model. The users of a cloud service do not own the computing infrastructure, but rent resources on demand in a pay-as-you-go model. This computing model provides numerous benefits both for the users

and the cloud providers. The users do not need to worry about maintenance or hardware upgrades to fulfill their computing needs, because the provider of the cloud service takes care of these details. Moreover, when users need more resources (e.g. storage space or CPU power) they can pay more and get access to the additional resources instantly for as long as needed. For the providers, a major benefit is that they have full control of the infrastructure which makes it easier to maintain, consolidate and effectively drive down costs while offering a service which is of higher quality. Consider the example where a database company sells databases to clients on premises versus renting the databases that are hosted by the database company itself. In the case where the database is installed on customer premises, different customers may use different database versions on different hardware with different configurations. In case of a fault there are multiple parameters that may be responsible and all these parameters may be different for each customer. When the databases are hosted by the database company and rented out to customers, the database company can keep a better control of all the different parameters that can affect the quality of the offered services.

Cloud services can be different in nature and are commonly referred to with the *XaaS* notation. XaaS stands for *X-as-a-Service*, where *X* is substituted with the service type that is being served by the cloud. The three most common cloud offerings are: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [73, 74].

- *SaaS* refers to different kinds of software running over the network, usually served as a web application to the client. All the processing is done centrally in a server, and the results are sent to the client. Minimum, or no configuration is needed by the client. Example SaaS services: Hosted e-mail, hosted storage (e.g. Dropbox), web sites.

- *PaaS* serves different kinds of platforms that other applications can be built upon. The clients get lower level access in the cloud stack and by using PaaS, one could deliver SaaS applications.
  Example PaaS services: Amazon Elastic Beanstalk, Google App Engine.

- *IaaS* delivers infrastructure on demand and is the type of cloud service that concerns this thesis. Clients get access to hardware and have similar access rights as if the hardware was hosted locally. The most common way that the infrastructure is delivered is in the form of VMs. IaaS provides OS level access to the clients. By using IaaS one could serve PaaS and SaaS applications, or do any other kind of computations that an ordinary computer would do.
  Example IaaS services: Amazon EC2, Google Compute Engine.

2.2.1 Infrastructure as a Service and Resource Consolidation

Cloud platforms become possible by the combination of numerous underlying technologies. Arguably, virtualization is the core technology that drives IaaS cloud platforms and provides many of the desired cloud features [2]. Some well known characteristics of cloud platforms

that would not be possible without virtualization are the elasticity and consolidation. The ability of a cloud to scale up or down its resources when needed is called elasticity. Efficient elasticity can provide better resource utilization and improve service performance while reducing energy consumption and costs both for the cloud tenants and the cloud providers. Consider a case where the tax authorities in a country have a few servers to host the government's online tax portal. The tax portal is mostly underutilized throughout the year since most people do not connect to the web site very frequently. Nonetheless, the costs are running as the servers consume energy resources and fixed floor space. There is a single day every year when the tax return due date is announced and people rush to file their tax return forms. This specific day the otherwise underutilized servers are not enough to meet the service demand and the portal crashes. So the tax office suffers from the downsides of *both worlds*: Overpaying for resources it doesn't need for most of the year, and when the critical time comes that the site must be up and running, the tax portal crashes. It turns out that this story is not fictional, but this is what repeatedly happens with the Norwegian online tax portal [75, 76]. If such a service was hosted in an elastic cloud service it would be possible to use less resources throughout the year (reduce the major costs), scale up the service a few hours before the tax return due is announced to be ready for the high demand, and scale down as needed when the demand decreases. Typically, the scaling is automated by specifying performance metric triggers [77].

The clouds are very dynamic in nature and the workload can be very unpredictable for the cloud providers. The cloud tenants are given the freedom to start and stop VMs at their own convenience so there are no guarantees about the expected next-minute utilization. Features such as the elasticity contribute to even greater unpredictability, as unexpected events may cause existing services to occupy more or less resources. This dynamicity leads to resource fragmentation [78], thus, cloud providers need to reconfigure their infrastructure in order to optimize resources and reduce costs and energy consumption. The most common way to achieve resource optimization is by using VM live migrations to consolidate fragmented resources. However, consolidation, especially when resources are overcommitted [79], needs to respect the Service Level Agreements (SLAs) between the cloud tenants and the provider. RQ 5 can be directly related to this issue: how could cloud providers increase infrastructure consolidation without violating the SLA? We provide a consolidation metric, the *delay* of workloads, as an answer to RQ 5 in Paper VI, and we perform bin packing of VMs based on the *delay*. The *delay* can be used as a directly observable metric in cloud environments where time-constrained applications are running.

Lots of research has revolved around mapping the problem of VM consolidation to the bin packing problem where Physical Machines (PMs) are assimilated to bins and VMs are assimilated to items. Generally, the aim is to reduce the number of PMs needed to pack different VMs across various resources such as CPU, memory, network I/O etc. Bin packing is known to be an NP hard problem, therefore, several heuristics have been applied when it comes to VM consolidation, including the well established First Fit Decreasing (FFD), Best Fit Decreasing (BFD) or variants of these algorithms [61, 80, 81, 82, 83, 84]. Other researchers have resorted to novel bio-inspired solutions, and perform the bin packing with algorithms such as the Ant Colony Optimization [85, 86].

The complexity of the consolidation increases exponentially as more dimensions are taken into account, hence, some studies reduce the problem to one dimensional bin packing by only considering the bottleneck resource. Similarly, Wood et al. [80] devised *Sandpiper*, a consolidation system that uses a simple formula to combine dimensions associated to different resources (CPU, memory and network) in one metric, called *volume*, in order to quantify the load of VMs and PMs and deploy a classic one dimensional FFD bin packing algorithm. The higher the utilization of a resource, the greater the *volume*. Sandpiper aspires to eliminate hotspots via pro-active VM migrations, where profiling is used to anticipate the occurrence of the overload. The consolidation is dynamic, i.e. whenever a hotspot is detected, VMs are migrating from the most overloaded PMs to the least loaded.

An example of another dynamic consolidation approach is presented in [87] where Beloglazov et al. propose to detect host overload using Markov chain modeling. The authors show that a necessary condition to improve the quality of VM consolidation is to maximize the mean time between migrations, and provide heuristics that trigger VM migrations while respecting this condition. They define the quality of VM consolidation to be inversely proportional to the mean number of active hosts over n time steps.

In [88], the authors present iPOEM, a consolidation system that tries to reduce energy consumption without violating SLA constraints. iPOEM tries to optimize two key parameters in a data center, the max CPU usage (CPU_{high}) a PM should tolerate before triggering a VM migration and the min CPU usage (CPU_{low}) for turning off a machine. The authors provide two theoretical results that are the core of the iPOEM algorithm: SLA violations can be reduced by reducing CPU_{high}, while the energy consumption can be decreased if CPU_{low} is increased. Based on these two intuitive observations, iPOEM performs a guided binary search to achieve two conflicting objectives, namely, reducing power consumption while operating within the SLA constraints.

Although most of the consolidation approaches use variants of deterministic bin packing algorithms, others use its stochastic counter part [89]. In the stochastic bin packing problem a list of items is given, where each item is a random variable. Moreover, an overflow probability p is provided. The goal is to pack the items into a minimum number of unit-bins of a given capacity such that the probability that the total size of the items in each bin exceeding 1 is at most p.

In order to take into account the dynamic nature of the resource consumption while still benefiting from the power of bin packing heuristics, some attempts in the literature have devised consolidation approaches that combine both ideas. For example, in [61], the authors used time series prediction techniques to predict the load, and then provided a probabilistic guarantee of satisfying the SLA constraints. Based on the forecast, the first-fit bin packing heuristic is used with a modified version where capacity is not a criteria, but the probability of exceeding the physical machine's capacity is.

Other interesting consolidation techniques are based on deep inspection of different workload characteristics. For example, in [90, 91], the authors attempt to characterize the workload and consolidate VMs with complementary resource usage over time. In [92], Wood et al.

inspect memory pages and collocate VMs with high memory sharing potential in order to take advantage of the memory page sharing features of hypervisors and increase memory density [93]. In [84], the authors include the network communication in their decision making process, and consolidate VMs to minimize inter-host traffic.

2.3 Lossless Interconnection Networks

A network can be *lossy* or *lossless* with regards to the delivery guarantees offered by the communication medium in the data link layer. In a lossy network there is no guarantee that an injected packet will arrive at its destination. Packets are forwarded even if switch buffers have been filled up due to congestion. In a case where a forwarded packet reaches a switch without available buffer space, the packet is simply dropped. The dropped packets have to be retransmitted. Conventional Ethernet is lossy[2]. In order to establish reliable communication streams over Ethernet the well known transport layer Transmission Control Protocol (TCP) is widely used. When TCP packets get dropped and no acknowledgments are received back to the sender after a specified timeout, the protocol interprets the packet loss as an indicator that congestion occurs and throttles down the packet injection rate [95]. In high performance systems this behavior is considered very costly. Packet retransmissions contribute both to increased communication latency, and unnecessary added overhead in the network as the retransmitted packets have to traverse the communication medium more than once. Typically, HPC environments employ lossless network technologies that prevent packets from being dropped, unless bit errors occur, by implementing link level flow control. When a network uses link level flow control, packets are not forwarded unless the sender knows that there is available buffer space in the receiver to store the packet in case of congestion. Inherently, all upper layer protocols run on top of a reliable link.

2.3.1 Deadlocks

The fact that packets are stored in a sender node and are not dropped or forwarded until there is available buffer space in the receiver node, introduces the potential for deadlocks if routing cycles (loops) exist [36], due to the First-In-First-Out (FIFO) nature of the buffers, as already pointed out in section 1.2. Deadlock is a state in which each member of a group of transactions is waiting for some other member to release a lock as defined by Coulouris et al. [96]. From a network point of view the *lock* that needs to be released is the equivalent to making buffer space available in the switches if there is none. A deadlock prevents packets from progressing towards their destination in the network, as all the switch buffers along the way have been filled and buffer space cannot be freed because some packets are mutually waiting for each other to proceed first.

As illustrated with an example in Fig. 2.2, four nodes, ①, ②, ③ and ④, that are connected

[2]The recently approved *IEEE 802.1Qbb* standard [94] defines lossless Priority-based Flow Control (PFC) for Ethernet on data center environments.

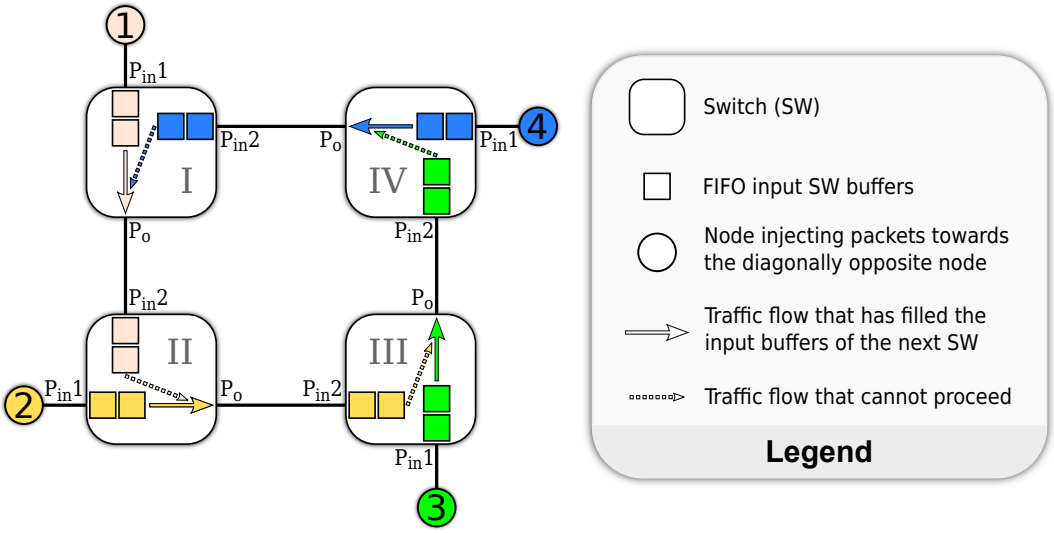


Fig. 2.2: A deadlocked network where the traffic is halted (no packet can proceed) as a result of a routing loop that led to all of the input FIFO switch buffers being filled.

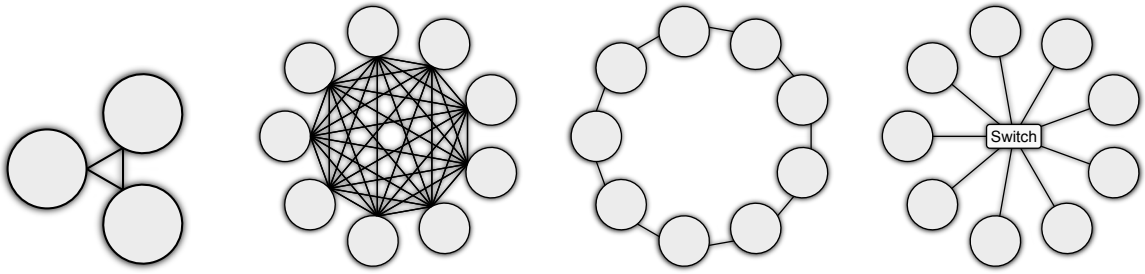
to the switches I, II, III and IV respectively, simultaneously inject packets in the network destined towards the node that is located diagonally from each corresponding node in the figure: ① → ③, ② → ④, ③ → ①, ④ → ②. The switches are equipped with FIFO buffers in the input ports (marked as $P_{in}X$) and when a packet arrives it is stored in a buffer and is only allowed to be forwarded through the output port (marked as P_o) when there is available buffer space in the next switch along the path that the packet has to follow[3]. In the case demonstrated by Fig. 2.2 all the switch buffers have been filled by packets that were sent by the node with the corresponding color, and the traffic is halted because no buffer can be freed to allow packets to progress towards their destination as indicated by the arrows. The routing algorithm is responsible to guarantee the deadlock freedom in a network. We discuss routing in section 2.3.3.

2.3.2 Network Topologies

The network topology defines how the network components, such as the switches and nodes, are arranged in the network. In general, the network topologies can be categorized into *direct* or *indirect* and *regular* or *irregular*. In direct topologies, end nodes are connected directly to each other, and depending on the topology the end nodes may be acting as forwarders of the traffic for other end nodes. In indirect topologies, switches are used to connect end nodes. A regular topology has a well defined structure for node and switch connections, while an irregular topology does not. Typically, the topologies are regular but note that a regular topology may become irregular if a node, switch, or link fails, or if different regular topologies get combined in a non-defined way. The network topology can affect the cost and performance of the network significantly.

In an ideal world, one would probably want to have a fully connected mesh topology where

[3]The path is determined by the routing function as we will see in section 2.3.3.



(a) Fully Connected Mesh - 3 Nodes, 3 Links

(b) Fully Connected Mesh - 9 Nodes, 36 Links

(c) Ring Topology

(d) Star Topology

Fig. 2.3: Fully Connected Mesh, Ring and Star topologies

each end node is directly connected to each other with a dedicated point-to-point link. No switches are involved. The diameter[4] of a fully connected mesh is 1. With such a network, the system is able to achieve maximum performance as there is no switching that adds extra latency, nor traffic contention as each node has dedicated resources to communicate with the others. This topology is known as a fully connected mesh. Nonetheless, as illustrated in Fig. 2.3(a) and Fig. 2.3(b), as the number of nodes scale up, the cost and complexity of the topology increases polynomially. When three nodes are fully connected only three links[5] are needed, while for nine nodes 36 links are needed. To generalize, the number of links needed in a fully connected mesh topology is $num\_links = num\_nodes \cdot (num\_nodes - 1)/2$. For a 40,000 nodes system, which is not uncommon to spot in the top 500 supercomputers list [33], one would need 799,980,000 links and twice as many network ports[6]. On the other hand, the ring and star topologies that are shown in Fig. 2.3(c) and Fig. 2.3(d) respectively, can be very cost effective, but again not scalable for different reasons. In the ring topology, where each node is connected only with two other nodes in a circular way that forms a ring, the nodes act as traffic forwarders for other nodes and the traffic flows in one direction. That gives a network diameter of $num\_nodes - 1$ to the ring topology and as the number of nodes increases, the network become slow. The star topology has a fixed network diameter of 2, but it is not feasible to build large port-count non-blocking switches[7] in order to scale to a high number of nodes, due to switch cost, size, and engineering complexity. Both the ring and star topologies also have a single point of failure; if any one node in the rings fails, the network will be disconnected, while if the switch in the star topology fails, the network will be disconnected as well.

The topologies in Fig. 2.3 are the simplest topologies that can be used to convey the potential issues and trade-offs, such as cost, performance and engineering or deployment complexity, that one has to take into account when designing or choosing to use a network

[4]The diameter of a network reveals the longest shortest-path between any two nodes in the network. Thus, when the diameter is short, the packets have to traverse fewer links and intermediate switches/nodes on average, before reaching to the destination.

[5]Assuming we talk about wired networks, each *link* translates into a *wire* in the physical world.

[6]Each link will always connect to two ports.

[7]A non-blocking switch is a switch that allows any input port to be connected to any free output port without affecting any of the existing connections [97].

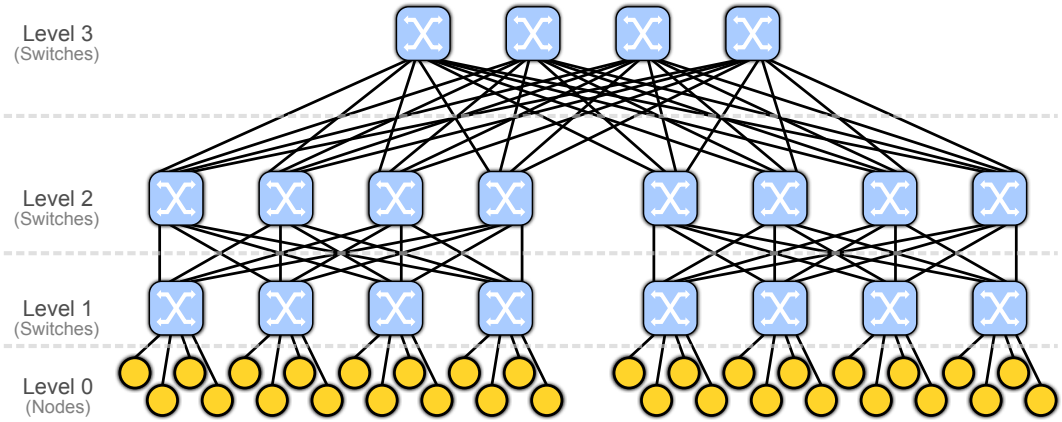


Fig. 2.4: An XGFT(3; 4,4,4; 1,4 4) Fat-Tree topology with 20 8-port switches and 32 nodes.

topology. In real life, more sophisticated topologies that balance the cost and performance while meeting the desired level of scalability are used. Some well known network topologies that are used in HPC clusters are the *Fat-Tree* [98], *Dragonfly* [99], $X-Dimension\ Mesh$ and $X-Dimension\ Torus$ [100] topologies.

The Fat-Tree Topology

In this thesis, and specifically in Paper III, Paper IV and Paper V, we have devised methods and algorithms that target particularly the Fat-Tree topology. The Fat-Tree is a scalable hierarchical network topology [98, 101], that is easy to build using commodity switches placed on different levels of the hierarchy [102]. The main idea behind the Fat-Trees is to employ *fatter* links between nodes, with more available bandwidth, towards the roots of the topology. The fatter links help to avoid congestion in the upper-level switches of the topology, and the bisection bandwidth is maintained[8]. Different variations of Fat-Trees are presented in the literature, including k-ary-n-trees [101], Extended Generalized Fat-Trees (XGFTs) [98], Parallel Ports Generalized Fat-Trees (PGFTs) and Real Life Fat-Trees (RLFTs) [104].

A k-ary-n-tree [101] is an n level Fat-Tree with k^n end nodes and $n \cdot k^{n-1}$ switches, each with $2 \cdot k$ ports. Each switch has an equal number of up and down connections in the tree, except for the root switches that have only down connections. The XGFT Fat-Tree extends the k-ary-n-trees by allowing both different number of up and down connections for the switches, and different number of connections at each level in the tree. The PGFT definition further broadens the XGFT topologies and permits multiple connections between switches. A large variety of topologies can be defined using XGFTs and PGFTs. However, for practical purposes, RLFTs, a restricted version of PGFTs, are introduced to define Fat-Trees commonly found in today's HPC clusters [105]. An RLFT uses the same port-count switches at all levels in the Fat-Tree.

[8]The *bisection bandwidth* of a network topology indicates the worst-case maximum bandwidth that can be achieved when the network is bisected (divided into two equally sized sets of nodes), and each node from the first set communicates only with one other node from the second set [103].

An XGFT that offers a *theoretical* full bisection bandwidth is illustrated in Fig. 2.4. A *theoretical* full bisection bandwidth indicates that there are enough links in the topology so that if the network is divided into two any equal sets of nodes, and each node from the first set communicates only with one other node from the second set, there should be no bandwidth degradation, i.e. there is no link sharing between all different flows. However, the *effective* bisection bandwidth will usually be less due to the effects of static routing as shown by Hoefler et. al in [103].

2.3.3 Routing

A routing algorithm defines the paths that packets have to follow to reach from any source to any destination in the network. Routing algorithms can be classified into *deterministic*, *oblivious* and *adaptive*. A deterministic algorithm will always choose the same path through the network for a packet with a given source and destination address without considering any other information such as the current network traffic. A deterministic routing algorithm guarantees ordered packet delivery in lossless networks. An oblivious routing algorithm may choose different paths for a given source and destination address, but similar to the deterministic routing, the chosen path is not dependent on the current status of the network. A decision can be made based on a scheme such as *random* or *round-robin* port selection. Note that deterministic routing is oblivious, but the opposite does not hold true. The adaptive algorithms take into account the current network status, and choose different paths for a given source and destination with the intention to avoid congested routes or faults. Note, however, that an adaptive routing algorithm may end up worsening congestion if there are no possible routes around a hot spot [106]. Adaptive and non-deterministic oblivious routing may introduce out-of-order packet delivery that might not be acceptable by some applications.

Furthermore, depending on where the routing decision is taking place, routing can be classified into *source routing* and *distributed routing* [97]. In source routing, the source node chooses the path that a packet will follow through the network and embeds the routing information in the packet header. In distributed (or destination-based) routing the packet header only carries the source and destination addresses. Each intermediate node (switch/router) decides where to forward each packet by looking up the destination address of the packet from a lookup table that is distributed to all of the switches/routers.

No matter if it is deterministic, oblivious, or adaptive, source or distributed, the routing algorithm is responsible to ensure the deadlock freedom in a network as briefly mentioned in section 2.3.1. Routing algorithms can be topology-aware [104, 105], that are optimized and target a specific network topology; or topology-agnostic [41, 107], that can route traffic even in irregular topologies or regular topologies that have been impacted by faults. A topology-aware routing algorithm will ordinarily be able to calculate the routing tables faster, and deliver higher performance for the given network topology, but on the contrary, the topology-agnostic routing algorithms can be fully fault tolerant and work for any given topology.

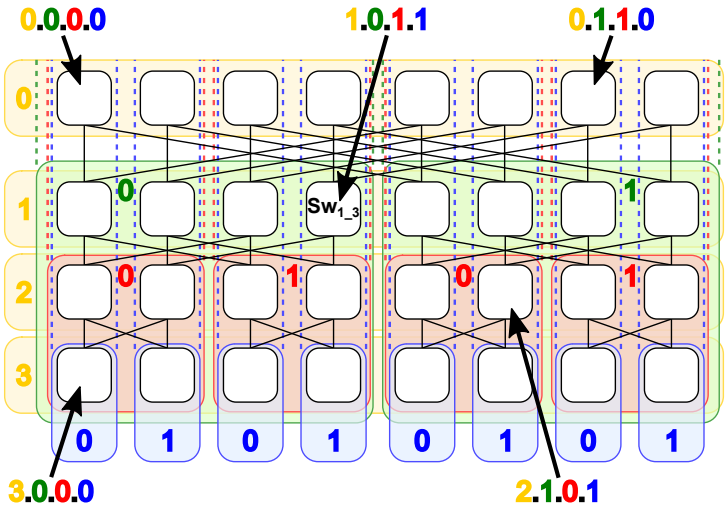


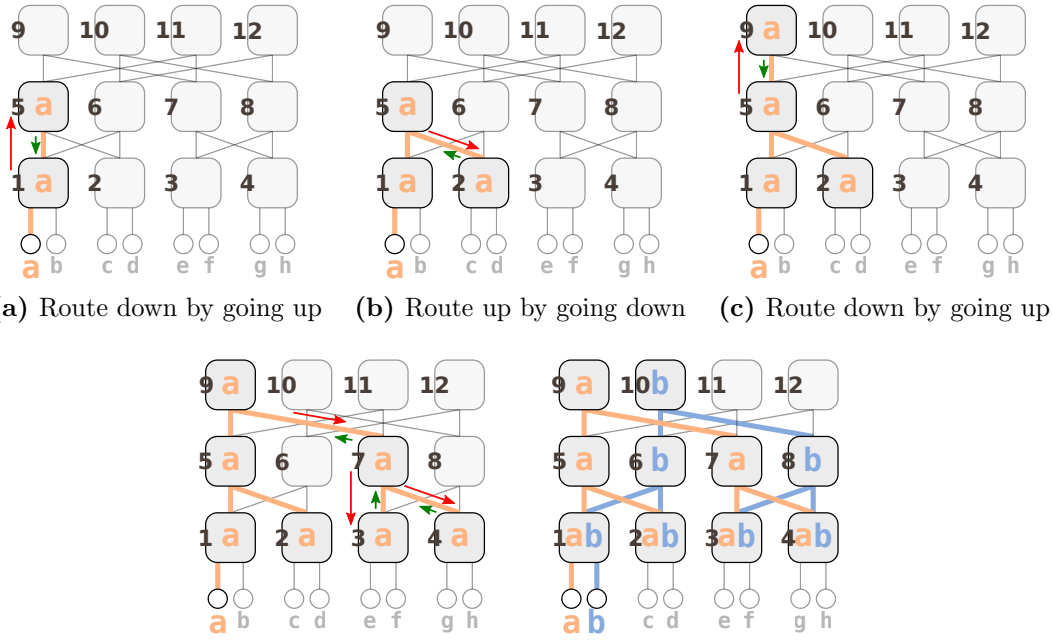
Fig. 2.5: *k*-ary-*n*-tree switch tuples assignment, where $k = 2, n = 4$. The nodes that are located at level 4 are omitted from this figure.

Fat-Tree Routing

The Fat-Tree routing algorithm (FTree) is a topology-aware routing algorithm for Fat-Tree topologies [104, 105, 108]. We have modified the Fat-Tree routing algorithm for the needs of Paper III, Paper IV and Paper V. In this section we are going to explain how the basic routing algorithm has been implemented in OpenSM[9]; FTree first discovers the network topology and each switch is marked with a tuple that identifies its location in the topology. Each tuple is a vector of values in the form of $(l, a_h, ..., a_1)$, where l represents the level where the switch is located. The a_h represents the switch index within the top-most sub-tree, and recursively the digits a_{h-1} until a_1 represent the index of the sub-tree within that first sub-tree and so on. For a Fat-Tree with n levels, the *root*-level (topmost or core) switches are located in level $l = 0$, whereas the leaf switches (where nodes are connected to), are located in level $l = n - 1$. The tuple assignment for an example *2*-ary-*4*-tree is shown in Fig. 2.5. Note that the implementation differs slightly from the theoretical definitions of different types of Fat-Trees where the *root*-level switches are located in level $l = n$, whereas the leaf switches are located in level $l = 1$ as illustrated in Fig. 2.4.

Once the tuples have been assigned, FTree iterates through each leaf-switch in an ascending tuple order, and for each downward switch port where nodes are connected in an ascending port-order the algorithm routes the selected nodes based on their LID. In Fig. 2.6 we show different phases of how routes towards node a are found. Switches in Fig. 2.6 are marked with numbers from $1 - 12$. FTree keeps port-usage counters for balancing the routes and starts by traversing the fabric upwards from the least loaded port while choosing the routes downwards, as shown in Fig. 2.6(a) with the red and green arrows respectively. In the first iteration all port counters are zero, so the first available upward port is chosen. For each level up, the newly reached switch (switch 5 in Fig. 2.6(a)) is selected as the switch to route all the traffic downwards towards the selected node (node a), from the incoming port which

[9]OpenSM is the SM software that is distributed with the OFED™ suite [52]. More information about subnet management is provided in section 2.4.2.



(a) Route down by going up **(b)** Route up by going down **(c)** Route down by going up

(d) Route up by going down **(e)** a and b nodes routed

Fig. 2.6: Fat-Tree (FTree) Routing Phases

through the switch was reached. Then the algorithm traverses the fabric downwards and assigns routes upwards towards that switch in a similar way as shown in Fig. 2.6(b). The same recursive operation continues until route entries for the selected node have been added to all of the necessary switches in the fabric, as depicted in Fig. 2.6(c) and 2.6(d). Next the algorithm will continue with the second node, node b in our case, as shown in 2.6(e), and so on until all nodes have been routed.

With the resulting routing in Fig. 2.6(e), if node f sends a packet towards a, path $P1$ through switches $3 \rightarrow 7 \rightarrow 9 \rightarrow 5 \rightarrow 1 \rightarrow a$ will be used, while path $P2$ through switches $3 \rightarrow 8 \rightarrow 10 \rightarrow 6 \rightarrow 1 \rightarrow b$ will be used if f sends a packet to b. Note that in Fig. 2.6(d) the routing towards node a has been completed, but there are some *blank* switches without routes towards node a; the switches $6, 8, 10, 11, 12$. In reality, FTree add routes in these *blank* switches as well. If a packet towards a arrives for example in switch 12, this switch knows that it has to forward the received packet down towards switch 6, while switch 6 knows that the received packet from 12 has to be forwarded to switch 1 to reach its destination a. However, the switches in the lower levels will never forward traffic towards node a to switch 12 because the routes upward will always push the packets towards switch 9. Note that the use of a single root switch per destination node counters the growth of wide congestion trees [106].

2.3.4 Network Reconfiguration

When faults occur in lossless networks the network has to be reconfigured to reroute the traffic that is affected by the fault. Another valid reason to trigger a network reconfiguration

is in the case where a routing algorithm is traffic-aware, i.e. the routing algorithm is taking into account the current traffic characteristics to improve performance [109]. If the traffic pattern changes, it is desired to reconfigure the network. Essentially, the outcome of a *network reconfiguration* is rerouting of the traffic in the network. Nevertheless, rerouting in lossless networks is not a trivial task due to the potential of introducing deadlocks. Traffic cannot be *just* redirected through another link because redirected traffic may introduce routing loops even if the initial routing was deadlock-free. To make things harder, even the coexistence of two deadlock-free routing functions, R_{old} and R_{new}, during the transition phase from the old to the new one, might not be deadlock free [110].

The types of reconfiguration can be categorized in *static* and *dynamic*. In static reconfiguration the traffic is halted and packets are drained from the network. Then the network is reconfigured with a new routing function R_{new}, before the traffic is resumed. Naturally, a static reconfiguration imposes system downtime, as during the reconfiguration process no traffic is flowing. However, static reconfiguration ensures deadlock freedom as there is no moment that two different routing functions, the R_{old} and R_{new}, coexist. As the time it takes to reconfigure the network can range in the order of minutes, especially in large subnets as we and others show in Paper V and [40, 41], dynamic reconfiguration is generally preferred over the static. In dynamic reconfiguration the traffic keeps flowing while the network gets reconfigured, but the reconfiguration must be done carefully in order to ensure the deadlock freedom of the network during the transition from R_{old} to R_{new} when the new routes are distributed to all switches.

Plenty of research exists in the domain of dynamic reconfiguration of lossless networks. Zafar et al. [111] discuss the tools and applicable methods on the IB architecture, that would allow the implementation of the *Double Scheme* [112] reconfiguration method. *Double Scheme* is using Virtual Layers (VLs)[10] to separate the new and the old routing functions. Lysne et al. [113] use a token that is propagated through the network to mark a reconfiguration event. Before the token arrives at a switch, traffic is routed with the old routing algorithm. After the token has arrived and been forwarded through the output ports of the switch, the traffic is flowing with the new routing algorithm. The *Skyline* by Lysne et al. [114], speeds up the reconfiguration process by providing a method for identifying the minimum part of the network that needs to be reconfigured. Sem-Jacobsen et al. [115] use the channel dependency graph to create a channel list that is rearranged when traffic needs to be rerouted. The rearranging is happening in such a way, that no deadlocks can occur. Robles-Gómez et al. [116] use close up*/down* graphs to compute a new routing algorithm which is close to the old one, and guarantees that the combination of old and new routing during transition does not allow deadlocks to be introduced. Bermúdez et al. [117] are concerned with the long time it takes to compute optimal routing tables in

[10]VLs provide the means to support independent data streams on the same physical link by using independent buffers in the switches. VLs are typically used for QoS, as well as by routing algorithms to resolve routing loops, that may lead to deadlocks, and improve performance, by reducing the Head-of-Line Blocking (HoLB) [41, 107]. HoLB is a common phenomenon that appears when the network is congested, and packets that could otherwise proceed towards their destination have to wait in the switch buffer queues behind packets that cannot proceed due to their contribution in the congestion. For more information regarding HoLB one can consult Gran's doctoral dissertation on congestion management [106].

large networks and the corresponding delay in the subnet becoming operational. They use some quickly calculated, but not optimal, provisional routes, and calculate the optimal routes *offline*. Since the provisional and the optimal routes are calculated based on the same acyclic graph, deadlock freedom is guaranteed. Guay, in his doctoral dissertation, provided insight into different aspects where dynamic reconfigurations are necessary [118]; First he looked into reconfiguration in then context of fault-tolerant environments, then into dynamic reconfiguration in the context of improving performance by reducing HoLB, and last, in the context of virtualization and VM live migration.

Part of this thesis is concerned with dynamic reconfiguration in the context of dynamic cloud environments. In cloud environments VMs can live migrate, thus a network reconfiguration is necessary. Moreover, the network traffic patterns continuously change as a result of either VMs migrating, or cloud tenants starting and stopping workloads that the cloud provider has no prior indication about. Hence performance-driven reconfigurations are desirable.

2.4 InfiniBand

IB is an industry standard, switched point-to-point lossless interconnect architecture developed by the InfiniBand™ Trade Association (IBTA). As of June 2017, IB is one of the most popular interconnects of choice in the list of top 500 supercomputers and has been used in 35.4% of the installations. IB is the interconnection network technology that has been used throughout this thesis to demonstrate our devised algorithms and prototypes that answer the research questions we asked in section 1.2. The primary issues we target in the work we present in Papers I, II, III, IV and V are centered on lossless networks in dynamic environments and in this section we provide background for core lossless network design characteristics, such as the addressing schemes, subnet management, high performance IOV, and how these components are related.

2.4.1 InfiniBand Addressing Schemes

IB is using three different types of addresses [1]. First is the LID which is a 16 bits long, layer two address. At least one unique LID is assigned to each end port or switch by the SM. As *end ports*, we refer to the terminal ports of the network, usually located on a network adapter. In IB a network adapter is called an HCA. The Local Identifiers (LIDs) are used to route traffic only within one subnet and are the base addresses that are used by the routing algorithm to calculate deadlock-free routes. Since the LID is 16 bits long, 65536 unique address combinations can be made, of which only 49151 (0x0001-0xBFFF) can be used as unicast addresses assigned to end ports or switches, while the remaining LIDs are reserved for multicast addresses. Consequently, the number of the available unicast addresses defines the maximum size of an IB subnet.

Second is the Global Unique Identifier (GUID) which is a 64 bits identifier assigned by the

manufacturer to each device (HCAs, switches, routers etc) and each end port. The SM may assign additional GUID addresses to an end port or a router port, which is particularly useful when SR-IOV VFs are enabled. Similar to the LID, the GUID must be unique in a subnet.

Third is the Global Identifier (GID). The GID is a 128 bits valid IPv6 unicast layer three address, and at least one is assigned to the end ports or multicast groups. The first GID is formed by combining a globally unique 64 bits subnet prefix that is assigned by the SM, and the GUID address of each end port. The GID provides a globally unique address for inter-subnet routing.

2.4.2 Subnet Management

An SM entity is running in one of the nodes in an IB fabric, and is responsible for the discovery and administration of the subnet [1]. The SM assigns LID addresses to the HCA ports and switches, calculates deadlock-free routes between all possible communicating pairs, and distributes the corresponding Linear Forwarding Tables (LFTs) to each switch in the fabric. IB employs destination-based routing, so the information distributed in the LFTs is a mapping of destination LIDs to corresponding output ports used for forwarding at the switches. Once the LFTs have been distributed the network is operational, while the SM periodically sweeps the fabric for faults/changes and serves as a path-characteristics resolution service (e.g. to resolve the supported MTU and speed that two communicating peers can use). A reconfiguration can be triggered when a fault or change is detected by a sweep. OpenFabrics' OpenSM is the most popular SM used on IB subnets, and the one that has been used throughout this work.

When the size of the fabric grows, the number of possible communication pairs increase polynomially. Consequently, the path computation in the SM increases polynomially as well, and depending on the topology and routing algorithm, the path computation can be in the order of several minutes. This is one of the scalability issues that becomes exceptionally important in dynamic cloud environments where reconfigurations may need to be triggered often. This issue is in particular addressed in Paper III for the Fat-Tree topology.

Note that even lossless network technologies that compete with IB still operate on the same principles and much of the terminology is similar or identical to what we present in this thesis. As an example, in Intel Omni-Path Architecture, a direct IB competitor, the equivalent of an IB HCA is called Host Fabric Interface (HFI) and the SM is called Fabric Manager (FM). However, the different layer-two/layer-three addresses, such as the LID, GUID, GID and other terms, are identically named [1, 37].

2.4.3 InfiniBand, SR-IOV, and Live Migrations in the Cloud

Performance is the major factor on HPC clusters. Providing elastic HPC on demand to customers over a cloud service is a challenging task due to the added overhead of

virtualization. When it comes to the interconnect network, none of the software emulation approaches discussed in section 2.1.3 are suitable for a virtual HPC environment. High performance interconnect network solutions delegate much of the workload into the hardware. In order to efficiently reduce latency and increase performance, the protocol stacks and the kernel of an OS are bypassed [119]. With the kernel bypassing in mind, the only suitable options to provide high performance networking in VMs are the direct device assignment techniques. For the aforementioned reasons, we, and other works related to IB and virtualization [30, 29, 38] chose to use SR-IOV for our experiments.

However, direct device assignment techniques pose a barrier for cloud providers if they want to use transparent live migrations that are notably useful for data center optimization. The essence of live migration is that the memory contents of the virtual machine are copied over to the remote hypervisor. Then the virtual machine is paused at the source hypervisor, and its operation is resumed at the destination as explained more thoroughly in section 2.1.1. When using software emulated IOV methods, the network interfaces are virtual so their internal state is stored into the memory and gets copied as well, thus, the downtime is in the order of a few milliseconds [60]. In the case of direct device assignment and SR-IOV VFs, the internal state of the network interface cannot be copied because it is stored into the hardware that is not able to move [31, 120]. The SR-IOV Virtual Function (VF) assigned to the VM will first need to be detached, the live migration will run, and a new VF will be attached at the destination. The described process will introduce some inescapable downtime which is in the order of several seconds.

In the case of a virtualized environment that is using SR-IOV InfiniBand VFs, the downtime of Live migrations will be even greater as we show in Paper I. When a VM that is using an SR-IOV IB VF is live migrated, the downtime will last from the moment that the VF is detached from the VM at the source hypervisor, until the moment a new VF is reattached at the destination. Moreover, some additional downtime and a measurable impact on the underlying network fabric and the SM will be introduced. The additional downtime and added overhead to the SM is due to the change of all three addresses of the IB SR-IOV VF [38]. The LID will change because the VM will be moved to a different physical host that is using a different LID and in current generation HCAs the LID of VMs is shared with the host LID [39]. The virtual GUID (vGUID) that is assigned by the SM to the source VF will change as well, because a different VF will be attached at the destination. Subsequently, since the vGUID is used to form the primary GID and the vGUID has changed, the GID will change too. The running applications and OS in the VM will suddenly be exposed to a new set of addresses at the destination, and all of the peer VMs will start sending path record queries to the SM while trying to reestablish the lost connectivity. These queries are causing supplementary downtime, and more significantly, extra overhead to the SM. If many migrations take place within a rather short time frame in a large data center, or if migrated nodes are communicating with many other nodes in the network, the SM can become a bottleneck since it will not be able to respond in a timely manner.

Other works in the context of virtualized HPC clouds with lossless interconnects are mostly limited to pointing out existing issues and demonstrating that the technology

has matured enough and virtualization overheads have sufficiently been reduced to make such clouds feasible [29, 30, 31, 32]. Others have focused on cloud tenant isolation at the link level [121, 122] while Ranadive, in his doctoral dissertation dedicated his efforts on virtualized resource management with lossless networks [123]. In particular, because the hypervisor is usually bypassed with high performance network technologies and SR-IOV as we explained in the first paragraph of this section, it is not as simple to monitor the resource usage and performance of the VMs. Moreover, HPC applications are latency sensitive. Ranadive considered those aspects, and developed tools and methods for monitoring and latency-aware scheduling of the VMs. Nonetheless, the closest related work to ours, that focuses in scalability and reconfiguration of lossless networks while live migrations are part of a cloud data center, is that of Guay [38, 120]. In [38], Guay et al. present a signaling mechanism that takes into account that all three IB addresses change when a VM is migrating, and notify the peers of the migrated VM to update their cached path information. In our work, and specifically in Paper I, we first show that alternatively if one has the possibility to migrate all three IB addresses, no signaling is needed. Then, in Paper II and Paper V we propose the IB SR-IOV vSwitch architecture that allows a VM to be migrated while carrying to the destination all of its associated addresses. In [120], Guay et al. also present a prototype that would allow VMs with IB SR-IOV VFs to migrate transparently[11]. This work is concentrating to IB internals, and the presented prototype migrates together with the VM the necessary hardware state of an IB VF so that the migration can happen in an application transparent way.

2.5 Simulators Used in this Thesis

In this section we clarify the scope of the three different simulators that have been used in this work. *Ibsim*, *ORCS* and *vSwitchMigrationSim*.

2.5.1 Infiniband Fabric Simulator

The Infiniband Fabric Simulator, or ibsim, is distributed as part of the OFED™ [52] software and could be best described as being an emulator. As specified in the *readme* file that is deployed with the tool, *ibsim emulates the fabric behavior by using MAD*[12] *communication with the SM/SA*[13] *and the PerfMgr. This simple tool is ideally suitable for various research,*

[11] Currently, because an SR-IOV VF has to be detached from the VM at the source hypervisor and reattached at the destination, all communications will be interrupted. Unless the application can handle such an interruption or an upper level protocol is used to ensure smooth communication resuming, like the Reliable Datagram Socket (RDS) protocol [124] that we also use in Paper I, the application will be disturbed.

[12] The Management Datagram (MAD) is a type of packet that IB uses, as the name implies, for managing the network.

[13] The Subnet Administration (SA) is a database that is built by the SM to store and serve different information that is needed for the operation of the subnet. As an example, when a host asks the SM for the path characteristics before communicating with another host, this information is served by the SM after an "SA Path Record" query has been received.

development, debug and testing tasks where IB subnet management is involved.

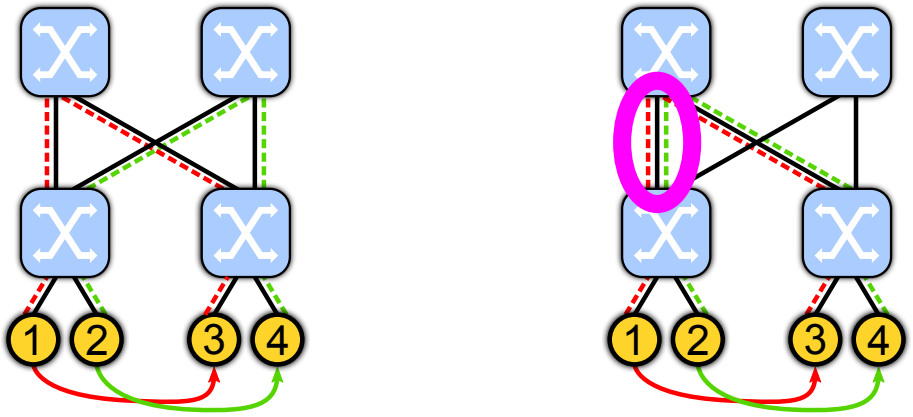
Ibsim works by reading a valid IB topology file, that can either be extracted from a real IB topology with the *ibnetdiscover* [125] command or generated in the same format, and emulates the behavior of the loaded subnet. Ibsim is an invaluable tool for testing new routing algorithms and study the behavior of the SM with different topologies easily, without one having to get access to a real system. An important detail to note is that by using ibsim, the performance of the SM will reflect reality if one is testing only internal elements of the SM. That is due to the fact that once the SM discovers the network (which can either be real or emulated by ibsim) and builds the SA database, many SM tasks are taking place internally and do not need any more network interaction. For example, once the subnet has been discovered, the routing algorithm runs locally. This means that the time it takes to route the network with a given routing algorithm is only affected by the local hardware that the SM is running on, even if the subnet is emulated by ibsim.

2.5.2 Oblivious Routing Congestion Simulator

The Oblivious Routing Congestion Simulator (ORCS) [51] is capable of simulating a variety of communication patterns on statically routed networks with the intent to study the effect of congestion. The simulator loads a network topology and the routing tables for all switches in the topology, and for each simulation the simulator runs a communication pattern in *turns* that in the simulator's language are called *levels*. Each *level* defines which nodes communicate with each other, and the results for each level are isolated from each other, i.e. one level has to finish before the next level is executed. In the end of the simulation, ORCS provides different metrics. For example, the *hist_max_cong* metric examines every single route used by any sender/receiver pair in a pattern and saves the maximal congestion along every route in a histogram, regardless of the level where the maximal congestion for each flow occurred in. Another example is the *hist_acc_band* metric. The *hist_acc_band* metric treats all connections in all levels equally, determines the congestion factor (a number between zero and one) for each flow and computes the fraction of peak bandwidth experienced by all the connections in one simulation run. Every simulation run results in one number, the fraction of peak bandwidth for this particular run. These results are stored in a histogram when multiple simulations are executed, and reported at the end.

If we have a network topology as shown in Fig. 2.7 with a pattern where node ① sends traffic to node ③ and node ② sends traffic to node ④, then the *hist_max_cong* metric would report 0 for the case depicted in Fig. 2.7(a) while the *hist_acc_band* metric would report 1.0 since there is no link sharing for the two flows. If the same network was routed differently as illustrated in Fig. 2.7(b), *hist_max_cong* would report 2 while the *hist_acc_band* would report 0.5 since the two flows share at least one link (in this case all switch to switch links are shared, but one shared link would be enough to provide these results).

In this thesis we used the ORCS simulator in Paper III, Paper IV and Paper V to evaluate our devised routing algorithms and reconfiguration methods. In all of these works we first used ibsim and OpenSM to generate and extract the routing tables from large emulated



(a) No congestion. Each flow is using dedicated links all the way for the communication. 100% of peak performance per flow.

(b) The two flows share at least one link. The first link where the flows meet from the source to the destination is the link that is highlighted with the oval shape. 50% of peak performance per flow.

Fig. 2.7: Two flows ($1 \rightarrow 3$ and $2 \rightarrow 4$) that are routed differently in the same topology.

topologies, and then we fed this information in ORCS to execute the simulations. As part of Paper III we also made significant contributions back to the ORCS project. In particular, we fixed several bugs that led to crashes or limited the generation of more complex traffic patterns, improved the support for parallel Message Passing Interface (MPI) execution of the simulator, and most significantly we overhauled the framework that is provided by the simulator to allow users to create new dynamic complex traffic patterns that can be tuned with command line arguments. All the git commits in the ORCS repository between 456e4b9c and d8b01f77 have been contributed by the author of this thesis[14].

2.5.3 Virtual Switch Migration Simulator

The Virtual Switch Migration Simulator (vSwitchMigrationSim) is a simulator that we created for the needs of the evaluation in Paper V. The purpose of vSwitchMigrationSim is to simulate an IB subnet with the SR-IOV vSwitch architecture that we proposed in Paper II, where VMs live migrate and their LID addresses are changing positions in the subnet. The ultimate goal for us was to use vSwitchMigrationSim to perform VM migrations and study the performance impact, to the routing quality, of our vSwitch reconfiguration methods that we presented in Paper II and Paper V. Then compare the results with the performance of an *optimally* routed subnet that has been routed with the *vSwitchFatTree* routing algorithm that we presented in Paper V as well. In the context of the vSwitch architecture, a quick, but not necessarily optimal, network reconfiguration that will result in minimal network disturbance must be triggered each time a VM migrates, to reroute the traffic heading to the migrated VM towards the destination hypervisor.

The vSwitchMigrationSim initially loads a network topology file and the corresponding

[14]https://github.com/cyberang3l/ORCS/compare/456e4b9c...d8b01f77
https://github.com/tim0s/ORCS/compare/456e4b9c...d8b01f77

routing tables that are extracted from the SM. Ibsim is used for the first step. vSwitch-MigrationSim then performs LID migrations (that imitate a live migration in a real life scenario where vSwitches would have been deployed) based on a chosen migration pattern, reconfigures the network as necessary based on our reconfiguration algorithms, and extracts the updated topology, routing tables as well as additional information such as how many switches were reconfigured after each migration. The extracted topology and routing tables are then loaded in the ORCS simulator to study the impact of the reconfiguration algorithms on the routing quality.

Four migration patterns have been implemented: *random*, *consolidate*, *grouping*, and *to-uniform*. In the *random* migration pattern, the user chooses how many random migrations must be performed. In a random migration a VM is selected randomly in the topology, and is migrated to a different randomly chosen hypervisor with available SR-IOV VFs. The *consolidate* migration pattern will initiate as many migrations as needed to pack all the VMs to the leftmost side of the Fat-Tree with the intent to use the least number of hypervisors possible that can accommodate the number of booted VMs. The VMs are migrated in a decreasing order, starting from the VM that is connected to the rightmost part of the Fat-Tree topology. Each VM is migrated to the leftmost part of the Fat-Tree where there is an available SR-IOV VF for the VM to connect to. In the *grouping* pattern, the user chooses a number of groups (a group can be seen as a cloud tenant) that the VMs will be grouped to. The VMs are chosen randomly to be equally distributed in the groups, and the pattern consolidates and isolates from other groups the VMs that belong to the same group, i.e. a hypervisor is not allowed to host VMs that belong to different groups. Last, the *to-uniform* migration pattern achieves a uniform VM distribution on the vSwitches in the network i.e. spreads out the VMs equally to all hypervisors. Note that the migration patterns in vSwitchMigrationSim are only trying to reflect potential scenarios that a cloud provider would be interested in. The implemented patterns are not designed to be optimal (do not intend to minimize the number of migrations), as what we care to study with this simulator is not the bin packing methods for different scenarios, but the resulting routing after a set of migrations have taken place.

Chapter 3

Summary of Research Papers

In this chapter we present an extended abstract for each of the papers that have been produced as part of this thesis, and we bind each paper to the research questions that have been presented in section 1.2. For a quick visual overview of the "*RQ* $\leftrightarrow$ *Paper*" mapping, the reader can consult Fig. 1.3. A short section also discusses the patents that have been produced as part of this work.

3.1 Papers

Paper I: A Novel Query Caching Scheme for Dynamic InfiniBand Subnets

Paper I addresses RQ 1. In large IB subnets the centralized SM is a potential bottleneck. When an IB subnet grows in size, the number of paths between hosts increases polynomially and the SM may not be able to serve the network in a timely manner when many concurrent path resolution requests are received. This scalability challenge is further amplified in dynamic virtualized cloud environments. When a VM with IB interconnect live migrates, the VM addresses change with the current *shared port* SR-IOV implementation [39]. These address changes result in additional load to the SM, as communicating peers send SA path record queries to the SM to resolve new path characteristics.

In this paper we first benchmark OpenSM to empirically demonstrate the SM scalability problems. Although the SM scalability is a known issue to the HPC community and the primary reason behind the work presented in this paper as well as the community's initiative to start a Scalable SA project [126], to the best of our knowledge there is no prior work that points out the issue with evidence. Then, after the benchmarking, we show that it is possible to significantly reduce the load towards the SM by introducing a novel SA Path Record Query caching scheme. In particular, we show that only a single initial SA path query is needed per communicating peer, independent of any subsequent (re)connection attempts. For the distinct case of VM migration, a prerequisite to make the cache work is that the migrated VMs should be able to carry their associated addresses to the destination hypervisor. Since live migration of VMs with IB SR-IOV VFs is not a production-ready

feature yet and a migration will break the communication of upper level applications[1], we used the RDS protocol to implement our prototype in this paper. The RDS protocol has built-in reconnection functionality in the case when the communication is interrupted, thus, the reconnection is transparent to the upper level application that is using RDS.

Paper II: Towards the InfiniBand SR-IOV Architecture

Paper II addresses RQ 2 and RQ 3 and builds on top of the results and observations of Paper I. In Paper I we showed that if a VM migrates with its associated IB addresses, the scalability of the SM can be greatly increased. Yet, when designing an SR-IOV architecture that would allow VMs to migrate together with their associated addresses, the network should be reconfigured after each migration. Network reconfiguration is another costly and complex task for the SM, and a new SR-IOV architecture that is designed to solve one scalability issue should take into account to not introduce a new one. Moreover, due to each VM having a dedicated layer-two address (recall that the LID is limited to 16 bits in IB) scalability issues in the space-domain of the subnet should be considered.

In this paper, we propose and analyze an SR-IOV virtual switch (*vSwitch*) architecture for IB, that takes into consideration the different scalability aspects we pointed out in the previous paragraph. Furthermore, as network reconfiguration time is critical to make live-migration a practical option, we accompany our proposed architecture with a scalable and topology agnostic dynamic reconfiguration method. A prototype for the reconfiguration method has been implemented and tested using OpenSM. Our results show that we are able to significantly reduce the reconfiguration time as route recalculations are no longer needed[2], and even in large IB subnets, for certain scenarios, the number of reconfiguration Subnet Management Packets (SMPs) sent by the SM to update the LFTs of the switches can be reduced from several hundred thousand that would be needed by a traditional full reconfiguration, down to a single one.

Paper III: Fast Hybrid Network Reconfiguration for Large-Scale Lossless Interconnection Networks

RQ 4 is mainly addressed by Paper III. Reconfiguration of high performance lossless interconnection networks is a cumbersome and time-consuming task. For that reason reconfiguration in large networks are typically limited to situations where it is absolutely necessary, for instance when severe faults occur. On the other hand, due to the shared and dynamic nature of modern cloud infrastructures, performance-driven reconfigurations are necessary to ensure efficient utilization and reduced fragmentation of resources.

In this work we present a scheme that allows for fast reconfigurations, by limiting the task to sub-parts of the network that can benefit from a local reconfiguration. Moreover, the

[1]To the best of our knowledge, live migration of VMs with IB SR-IOV VFs has only been demonstrated by Guay et al. in [120] so far.

[2]The recalculation of routes is the most costly task in a reconfiguration scenario.

presented scheme is able to use different routing algorithms for different sub-parts within the same subnet. Hardware experiments and large scale simulation results show that we are able to significantly reduce reconfiguration time from 50% to as much as 98.7% for very large topologies, while improving performance. The proposed scheme can take advantage of the fact that large HPC systems and clouds are shared by multiple tenants running isolated tasks. In such scenarios tenant inter-communication is not allowed, thus the workload deployment and placement scheduler should try to avoid fragmentation to ensure efficient resource utilization. That is, the majority of the traffic per tenant can be contained within consolidated subparts of the network, subparts we can reconfigure in order to improve the overall performance. We also present a Fat-Tree routing algorithm that reconfigures a network given a user-provided node ordering. Having the ability to route a network with a given node order can be very effective, and one strategy we use to obtain better performance is to route nodes in the order of the amount of traffic each node receives. A simple way to determine the receiving traffic per node is to read the network port counters. In such a way the administrator doesn't even have to know details about the jobs executed by tenants (applications are treated as black-boxes), making this type of routing attractive for dynamic environments.

Paper IV: Compact Network Reconfiguration in Fat-Trees

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.

In this paper, we propose a novel routing algorithm for fat-tree topologies, SlimUpdate. SlimUpdate employs path preservation techniques to achieve a decrease of up to 80% in the number of total path modifications, as compared to the OpenSM's fat-tree routing algorithm. However, SlimUpdate can be slow due to its recursive programming nature, and it is impractical to use it as often as needed by cloud infrastructures. Thus, in Paper IV we also present a faster metabase-aided rerouting method for Fat-Trees, based on destination leaf-switch multipathing that addresses RQ 4 differently than the method we presented in Paper III. The metabase routing scheme will first calculate the different possible spanning trees for each destination leaf-switch, and store that information in an in-memory database. Then the compute nodes are routed based on the pre-calculated paths. Naturally, the initial routing calculation will be slower than usual as these two phases are involved. Nonetheless, when a subsequent reconfiguration is needed alternative paths that already exist in the database will be selected, instead of being calculated again, resulting in a quicker reconfiguration. Note that the logic of allocating calculated paths to the actual compute node destinations is up to the routing algorithm, and not a part of our proposed metabase-aided routing mechanism. As our results show, the metabase-aided routing saves up to 85% of the total routing time over a traditional rerouting scheme where routing paths have to be calculated when the reconfiguration is triggered.

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

In this paper, we build on our already proposed vSwitch SR-IOV architecture for IB. We first discuss in detail the space-domain scalability issues related to the layer-two LID addresses, and suggest potential solutions based on the IB specification. Then we discuss routing strategies for virtualized environments using vSwitches, and present an efficient routing algorithm for Fat-Trees, followed by a dynamic reconfiguration method that is designed to minimize imposed overhead on Fat-Tree topologies. Our results show significant reduction in the reconfiguration time when compared to a traditional full reconfiguration, as route recalculations can be eliminated, and for certain scenarios, the number of reconfiguration SMPs sent to switches is reduced from several hundred thousand down to a single one without degrading the routing quality. Moreover, the Fat-Tree topology-aware reconfiguration method presented in this paper reduces significantly the number of reconfiguration SMPs that need to be sent to switches when compared with the topology-agnostic method that we introduced together with the vSwitch architecture in Paper II. We perform an extensive performance evaluation of our reconfiguration method, and draw conclusions by deducing from empirical observations of real hardware behavior as well as large-scale simulations.

The first part of Paper V that discusses vSwitch space-domain scalability issues is IB specific. Other competing interconnects may suffer from similar issues if a vSwitch architecture for lossless networks is implemented[3], while others currently may not[4]. Nonetheless, the rest of the challenges presented in this paper, although demonstrated on IB, are issues related to the nature of vSwitch-powered lossless networks in the context of dynamic cloud environments that are using VM live migrations, or are in need for frequent performance-driven reconfigurations, and provide answers to RQ 3.

Paper VI: The Concept of Workload Delay as a Quality-of-Service Metric for Consolidated Cloud Environments with Deadline Requirements

Paper VI provides one approach to answer RQ 5. VM consolidation in the cloud has received significant research interest. A large body of approaches for VM consolidation in data centers resort to variants of the bin packing problem which tries to minimize the number of deployed physical machines while meeting the SLA constraints. The most common strategy to bin-pack VMs is to satisfy a level of utilization of the physical machines. However, it is not clear how the level of utilization is related to the performance of the VMs and the user experience for users that use the VMs. After all, the way that the degraded performance of long-running jobs (that are typical in cluster and Big Data deployments) is

[3]The Bull eXascale Interconnect (BXI), which is another direct competitor technology to IB, scales up to 64k nodes [35]. Although the architectural specifications of BXI are not publicly available, this is an indication that the layer two addresses in BXI are 16 bits long as is in IB, thus, our space-domain scalability suggestions may not be applied directly to other technologies, but are still relevant.

[4]Intel Omni-Path Architecture utilizes 24 bits long LIDs [34], thus, one can have larger subnets than IB.

manifested to the user, is in the form of a workload that took longer than expected to be completed, i.e. the workload is *delayed*.

In this paper we introduce the concept of workload *delay* as a QoS metric that captures directly the resulting degradation that a cloud user would experience in the case where the SLA is violated. We show that the delay can be accumulated in a complex way, and although different workloads may look identical in the long run, the delay can be manifested and impact each workload very differently when there exist minor differences in the short term. Our results, that are based on real-life trace-based simulations, show that when VMs are consolidated based on the level of utilization there is little control over the resulting delay. This is a particularly significant drawback when running jobs with deadline requirements. On the other hand, we are able to control the delay much better if we take into account our suggested metric of the *delay*.

Paper VI provides an initial approach to formulate and demonstrate the potential of the concept of delay with an example based on a real-life performance dataset. The consolidation results are not yet intended to demonstrate a competitive consolidation solution. That is, we provide directions for future work, and point out different aspects that should be considered in order to apply the concept of delay in real-life scenarios.

3.2 Patents

Five patent applications have been submitted directly from the results of this thesis. Four have already been published (not granted yet) [127, 128, 129, 130], whereas one is pending publication. The patents have been filed by Oracle Corporation, which was one of the industrial partners of the Efficient and Robust Architecture for the big data Cloud (ERAC) project [131], and the partner that we had the closest collaboration with. The ERAC project was the umbrella project that mainly funded the work for this doctoral dissertation.

Chapter 4

Closing Remarks

This thesis was mainly funded by the ERAC project, and the goal of the ERAC project was to provide, as the name of the project implies, an efficient and robust architecture for the emerging Big Data clouds. As such, we chose to address challenges in the context of high performance virtualized clouds, that are needed to process efficiently, demanding Big Data workloads. In the beginning of the project we started from the higher cloud levels, in the cloud orchestration, with the intention to follow a top-down approach, and tried to deploy an OpenStack[1] cloud with the XEN[2] hypervisor and IB. Soon we realized that none of the open source enterprise cloud platforms are ready to accommodate high performance lossless interconnects, and many of the features that are working *out-of-the-box* with Ethernet technology, are not necessarily working with high performance lossless network technologies. So instead, we followed a bottom-up approach as explained in the introductory chapter 1. Our initial efforts did not have a direct scientific outcome in the cloud orchestration layer, but still, we made community contributions back to the utilized projects by filing bug reports and fixing bugs whenever encountered[3]. Even a security vulnerability that got assigned a Common Vulnerabilities and Exposures (CVE) number was discovered and reported during this learning and experimentation period[4]. Additional contributions that are part of this doctoral dissertation include a presented poster [132] and two contributed talks in well established HPC community venues [133, 134].

Future Work

In this thesis we explored technologies and techniques that would allow cloud providers to enable efficient high performance cloud environments. As we presented in Fig. 1.1, cloud

[1] http://web.archive.org/web/20170516144646/https://en.wikipedia.org/wiki/OpenStack
[2] http://web.archive.org/web/20170516144751/https://en.wikipedia.org/wiki/Xen
[3] http://web.archive.org/web/20170516144015/https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=721345
http://web.archive.org/web/20170516144109/https://bugs.launchpad.net/nova/+bug/1201795
[4] http://web.archive.org/web/20170516144204/https://bugs.launchpad.net/nova/+bug/1202266
http://web.archive.org/web/20161228085607/http://lists.openstack.org/pipermail/openstack-announce/2013-November/000161.html

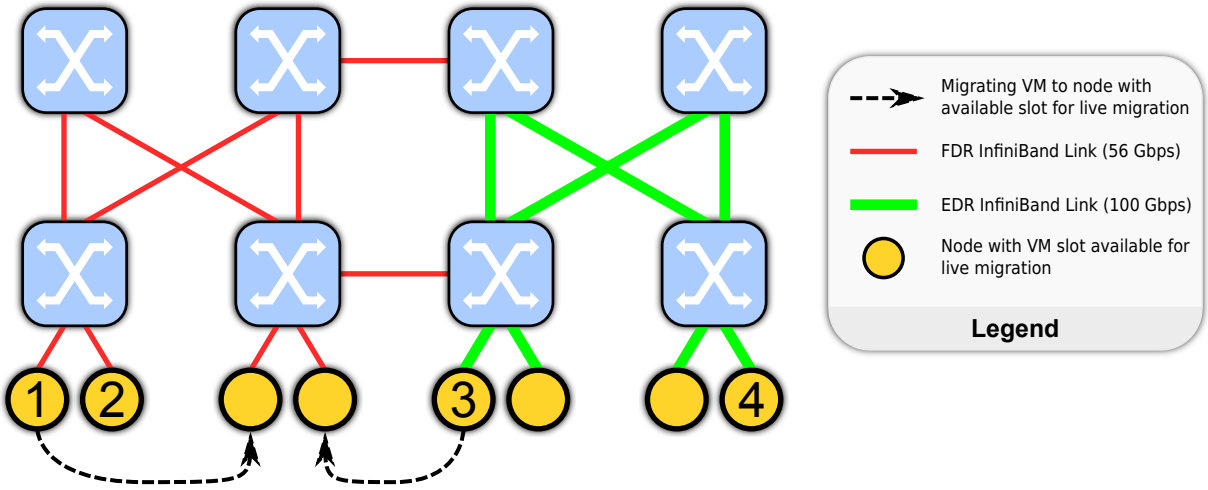


Fig. 4.1: An non-homogeneous IB topology that mixes FDR and EDR generation IB.

platforms are very complex and composed of several components and corresponding research objectives in different areas. In this work we spent most of our resources on improving the cloud infrastructure from the perspective of the network, by using lossless network technologies. The work was concerned with the dynamic nature of virtualized clouds and the cases where VMs need to be live migrated in order to improve performance and consolidation in the cloud. In this section we identify and present some future work.

In Paper I we presented an SA Path Record caching mechanism that would work in a vSwitch powered lossless network, but currently, for the caching method to work, the network must be homogeneous. Otherwise some migrations may break connectivity between hosts and the hosts that lost connectivity will start requesting the new path characteristics from the SM again. This behavior revives the problem we tried to solve in the first place by introducing the cache. Consider the simple case presented in Fig. 4.1 where an organization had a small cluster based on the FDR generation of IB. Recently they extended the cluster to accommodate new nodes but the newly acquired equipment is based on EDR IB which is newer and faster than FDR IB. In this scenario the network is not homogeneous anymore since different network speeds and features are supported in parts of the network. As such, if VM ① migrates to the available slot pointed by the dashed arrow, the cache will work as expected for all hosts. If on the other hand VM ③ migrates to the slot pointed by the corresponding dashed arrow, then VM ④ will not be able to communicate with VM ③ anymore. How would it be possible to allow location-oblivious migrations of vSwitch-powered VMs in a non homogeneous network without introducing new scalability issues in the SM?

For vSwitch migrations we did not study the deadlock possibilities when VMs are migrated and switches are reconfigured. For the moment, we suggested a partially static reconfiguration scheme to ensure deadlock freedom, a scheme that drops traffic only towards the migrated VM until the reconfiguration is completed, but not the rest of the network. Since at a certain point the VM is suspended and exhibits some downtime during the migration, a partially static reconfiguration is enough if completed in less time than the

downtime exhibited by the VM[5]. As we show in Paper V based on empirical measurements, the reconfiguration methods we presented are quicker than the migration downtime of VMs that is typically reported in real-life scenarios. But as the networks grow in size and live migration improvements lead to less VM downtime, in the future a partially static reconfiguration may not be enough and might lead to unnecessary packet drop that could be avoided if the reconfiguration was dynamic. A feasibility study of if, and how a dynamic reconfiguration can ensure deadlock freedom in the context of VM migration, is a possible future research direction for our proposed vSwitch architecture. Note that in existing work that has studied the reconfiguration of lossless networks, for example in the case when new nodes appear or faulty nodes disappear, nodes access the network with a unique identifier (LID) always from the same network location. In the case where we have dynamic subnets, the same node identifier may disappear from one part of the network, and reappear somewhere else as a result of a VM migration.

In Paper III we provided a topology-aware hybrid reconfiguration scheme. As a future direction our proposal would be to investigate how could one segment a topology agnostic network in a similar way as the Fat-Tree network is naturally segmented with sub-trees? How should a topology-agnostic routing algorithm route a segmented network to allow the re-routing of different network segments, with different routing algorithms, without affecting the performance or introducing deadlocks to the global routing? A next step would be to study the possibility of having hierarchical distributed subnet management with different SM workers being responsible for different network segments, and all workers reporting to a master SM in the subnet.

As for the concept of the delay that we introduced in Paper VI, a more advanced consolidation algorithm should be tested based on workload characterization and not just a simple best fit decreasing bin packing. So far we only demonstrated the concept of *delay* on time-shared CPU consumption data. The *delay* could also be used to capture the performance of different metrics such as latency, and bandwidth, and can even be affected by metrics that are not time-shared such as memory. For instance, if there is not enough memory in a system and the swap memory is used, this can have serious implications on the measured *delay*. Thus, additional performance parameters should be included in the calculation of the delay as future work.

[5]In any case, when the VM experiences downtime due to a migration, packets towards this VM will be discarded. So it does not matter if the packets will be discarded because of a static reconfiguration or because they reached the destination but the VM hasn't yet resumed its operations.

Papers

Paper I	**A Novel Query Caching Scheme for Dynamic InfiniBand Subnets** [135]
Authors	Evangelos Tasoulas, Ernst Gunnar Gran, Bjørn Dag Johnsen, Tor Skeie
Published at	IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) 2015
Paper II	**Towards the InfiniBand SR-IOV Architecture** [136]
Authors	Evangelos Tasoulas, Ernst Gunnar Gran, Bjørn Dag Johnsen, Kyrre Begnum, Tor Skeie
Published at	IEEE International Conference on Cluster Computing (IEEECluster) 2015
Paper III	**Fast Hybrid Network Reconfiguration for Large-Scale Lossless Interconnection Networks** [137]
Authors	Evangelos Tasoulas, Ernst Gunnar Gran, Tor Skeie, Bjørn Dag Johnsen
Published at	IEEE International Symposium on Network Computing and Applications (NCA) 2016
Paper IV	**Compact Network Reconfiguration in Fat-Trees** [138]
Authors	Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen, Tor Skeie, Evangelos Tasoulas
Published at	Springer Journal of Supercomputing in 2016
Paper V	**Efficient Routing and Reconfiguration in Virtualized HPC Environments with vSwitch-enabled Lossless Networks**
Authors	Evangelos Tasoulas, Feroz Zahid, Ernst Gunnar Gran, Kyrre Begnum, Bjørn Dag Johnsen, Tor Skeie
Submitted to	Wiley Journal of Concurrency and Computation: Practice and Experience
Paper VI	**The Concept of Workload Delay as a Quality-of-Service Metric for Consolidated Cloud Environments with Deadline Requirements**
Authors	Evangelos Tasoulas, Hugo Lewi Hammer, Hårek Haugerud, Anis Yazidi, Alfred Bratterud, Boning Feng
Published at	IEEE International Symposium on Network Computing and Applications (NCA) 2017

List of Acronyms

AWS Amazon Web Services.

BFD Best Fit Decreasing.
BXI Bull eXascale Interconnect.

CA Conceptual Analysis.
CA/M Conceptual Analysis/Mathematical.
CPU Central Processing Unit.
CS Computer Science.
CVE Common Vulnerabilities and Exposures.

ERAC Efficient and Robust Architecture for the big data Cloud.

FFD First Fit Decreasing.
FIFO First-In-First-Out.
FM Fabric Manager.

GID Global Identifier.
GUID Global Unique Identifier.

HCA Host Channel Adapter.
HCAs Host Channel Adapters.
HFI Host Fabric Interface.
HoLB Head-of-Line Blocking.
HPC High Performance Computing.

IaaS Infrastructure as a Service.
IB InfiniBand.
ibsim Infiniband Fabric Simulator.
IBTA InfiniBand™ Trade Association.
IDC International Data Corporation.
IoT Internet of Things.
IOV Input/Output Virtualization.
IT Information Technology.

LFTs Linear Forwarding Tables.
LID Local Identifier.
LIDs Local Identifiers.

MAD Management Datagram.
MAPE Monitor-Analyze-Plan-Execute.
MPI Message Passing Interface.
MTU Maximum Transmission Unit.

OFED™ OpenFabrics Enterprise Distribution.
ORCS Oblivious Routing Congestion Simulator.
OS Operating System.
OSs Operating Systems.

PaaS Platform as a Service.
PCI Peripheral Component Interconnect.
PF Physical Function.
PFC Priority-based Flow Control.
PMs Physical Machines.

QoS Quality of Service.

RDS Reliable Datagram Socket.
RQ Research Question.

SA Subnet Administration.
SaaS Software as a Service.
SAN Storage Area Network.
SLA Service Level Agreement.
SLAs Service Level Agreements.
SM Subnet Manager.
SMPs Subnet Management Packets.
SR-IOV Single-Root IOV.

TCP Transmission Control Protocol.

VF Virtual Function.
VFs Virtual Functions.
VLs Virtual Layers.
VM Virtual Machine.
VMM Virtual Machine Monitor.
VMs Virtual Machines.

Bibliography

[1] InfiniBand Trade Association. InfiniBand Architecture General Specifications 1.3, 2015.

[2] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer*, 45(12):65–72, 2012.

[3] International Data Corporation. 2016 IoT Mid-Year Review: A Report Card for Everyone, August 2016. Doc # WC20160804.

[4] Facebook Inc. Facebook's Top Open Data Problems. https://web.archive.org/web/20170510123250/https://research.fb.com/facebook-s-top-open-data-problems/. [Online; accessed 17-May-2017].

[5] Facebook Inc. Facebook statistics as of December 31, 2016. https://web.archive.org/web/20170306004707/http://newsroom.fb.com/company-info/. [Online; accessed 17-May-2017].

[6] Google Inc. YouTube Engineering and Developers Blog: Machine learning for video transcoding. https://web.archive.org/web/20170106201643/https://youtube-eng.googleblog.com/2016/05/machine-learning-for-video-transcoding.html. [Online; accessed 17-May-2017].

[7] International Data Corporation. The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth the in the Far East, December 2012.

[8] M. Chen, S. Mao, and Y. Liu. Big Data: A Survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.

[9] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan. Computational Solutions to Large-scale Data Management and Analysis. *Nature Reviews Genetics*, 11(9):647–657, 2010.

[10] S. Matsuoka, H. Sato, O. Tatebe, M. Koibuchi, I. Fujiwara, S. Suzuki, M. Kakuta, T. Ishida, Y. Akiyama, T. Suzumura, K. Ueno, H. Kanezashi, and T. Miyoshi. Extreme Big Data (EBD): Next Generation Big Data Infrastructure Technologies Towards Yottabyte/Year. *Supercomputing frontiers and innovations*, 1(2), 2014.

[11] J. L. Reyes-Ortiz, L. Oneto, and D. Anguita. Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf. *Procedia Computer Science*, 53:121 – 130, 2015.

[12] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5):14–22, September 2009.

[13] Amazon.com Inc. Amazon Web Services. https://aws.amazon.com. [Online; accessed 17-May-2017].

[14] Dropbox Inc. Scaling to Exabytes and Beyond. https://web.archive.org/web/20170128210506/https://blogs.dropbox.com/tech/2016/03/magic-pocket-infrastructure/. [Online; accessed 17-May-2017].

[15] Cisco Systems Inc. Cisco Global Cloud Index: Forecast and Methodology, 2015-2020, 2016.

[16] R. Jain and S. Paul. Network Virtualization and Software Defined Networking for Cloud Computing: A Survey. *IEEE Communications Magazine*, 51(11):24–31, November 2013.

[17] J. Michelsen and J. English. What Is Service Virtualization? In *Service Virtualization: Reality is Overrated*, pages 27–35. Apress, Berkeley, CA, 2012.

[18] W. Vogels. Beyond Server Consolidation. *Queue*, 6(1):20–26, January 2008.

[19] K. Hamdi and M. Kefi. Network-Aware Virtual Machine Placement in Cloud Data Centers: An Overview. In *International Conference on Industrial Informatics and Computer Systems (CIICS)*, pages 1–6, March 2016.

[20] A. Beloglazov and R. Buyya. Energy Efficient Allocation of Virtual Machines in Cloud Data Centers. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 577–578, May 2010.

[21] W. L. Guay, S. Reinemo, B. Johnsen, C. Yen, T. Skeie, O. Lysne, and O. Tørudbakken. Early Experiences with Live Migration of SR-IOV Enabled InfiniBand. *Journal of Parallel and Distributed Computing*, 78:39 – 52, 2015.

[22] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas. Performance Evaluation of Amazon EC2 for NASA HPC Applications. In *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date*, ScienceCloud '12, pages 41–50, New York, NY, USA, 2012. ACM.

[23] J. Vienne, J. Chen, M. Wasi-Ur-Rahman, N. S. Islam, H. Subramoni, and D. K. Panda. Performance Analysis and Evaluation of InfiniBand FDR and 40GigE RoCE on HPC and Cloud Computing Systems. In *IEEE 20th Annual Symposium on High-Performance Interconnects*, pages 48–55, August 2012.

[24] IBM Corporation. An Architectural Blueprint for Autonomic Computing (Fourth Edition), 2006.

[25] S. Singh and I. Chana. QoS-Aware Autonomic Resource Management in Cloud Computing: A Systematic Review. *ACM Comput. Surv.*, 48(3):42:1–42:46, December 2015.

[26] A. R. Hummaida, N. W. Paton, and R. Sakellariou. Adaptation in Cloud Resource Configuration: A Survey. *Journal of Cloud Computing*, 5(1):7, 2016.

[27] S. Singh and I. Chana. A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges. *Journal of Grid Computing*, 14(2):217–264, 2016.

[28] G. Aceto, A. Botta, W. Donato, and A. Pescapè. Cloud Monitoring: A Survey. *Computer Networks*, 57(9):2093 – 2115, 2013.

[29] M. Hillenbrand, V. Mauch, J. Stoess, K. Miller, and F. Bellosa. Virtual InfiniBand Clusters for HPC Clouds. In *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, CloudCP '12, pages 9:1–9:6, New York, NY, USA, 2012. ACM.

[30] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda. SR-IOV Support for Virtualization on InfiniBand Clusters: Early Experience. In *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 385–392, May 2013.

[31] V. Mauch, M. Kunze, and M. Hillenbrand. High Performance Cloud Computing. *Future Generation Computer Systems*, 29(6):1408–1416, 2013.

[32] P. Rad, R. V. Boppana, P. Lama, G. Berman, and M. Jamshidi. Low-latency Software Defined Network for High Performance Clouds. In *10th System of Systems Engineering Conference (SoSE)*, pages 486–491, May 2015.

[33] E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer. Top 500 The List. http://www.top500.org, 2017. [Online; accessed 17-May-2017].

[34] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak. Intel® Omni-path Architecture: Enabling Scalable, High Performance Fabrics. In *IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 1–9, August 2015.

[35] S. Derradji, T. Palfer-Sollier, J. P. Panziera, A. Poudes, and F. W. Atos. The BXI Interconnect Architecture. In *IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 18–25, August 2015.

[36] M. Karol, S. J. Golestani, and D. Lee. Prevention of Deadlocks and Livelocks in Lossless Backpressured Packet Networks. *IEEE/ACM Transactions on Networking*, 11(6):923–934, December 2003.

[37] A. Russell. The Intel® Omni-Path Architecture: Game-Changing Performance, Scalability, and Economics. In *Cray User Group (CUG) 2016*, May 2016. [Online; accessed 17-May-2017].

[38] W. L. Guay, S. A. Reinemo, B. D. Johnsen, T. Skeie, and O. Torudbakken. A Scalable Signalling Mechanism for VM Migration with SR-IOV over Infiniband. In *IEEE*

18th International Conference on Parallel and Distributed Systems, pages 384–391, December 2012.

[39] J. Morgenstein. Add SRIOV support for IB interfaces. https://web.archive.org/web/20170329185211/http://www.mail-archive.com/linux-rdma@vger.kernel.org/msg11956.html. [Online; accessed 17-May-2017].

[40] P. Vignéras and J. Quintin. The BXI Routing Architecture for Exascale Supercomputer. *The Journal of Supercomputing*, 72(12):4418–4437, 2016.

[41] J. Domke, T. Hoefler, and W. E. Nagel. Deadlock-Free Oblivious Routing for Arbitrary Topologies. In *2011 IEEE International Parallel Distributed Processing Symposium*, pages 616–627, May 2011.

[42] A. Iosup, N. Yigitbasi, and D. Epema. On the Performance Variability of Production Cloud Services. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 104–113, May 2011.

[43] Y. El-Khamra, H. Kim, S. Jha, and M. Parashar. Exploring the Performance Fluctuations of HPC Workloads on Clouds. In *IEEE Second International Conference on Cloud Computing Technology and Science*, pages 383–387, November 2010.

[44] O. Khalid, I. Maljevic, R. Anthony, M. Petridis, K. Parrott, and M. Schulz. Dynamic Scheduling of Virtual Machines Running HPC Workloads in Scientific Grids. In *3rd International Conference on New Technologies, Mobility and Security*, pages 1–5, December 2009.

[45] G. Dodig-Crnkovic. Scientific Methods in Computer Science. In *Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden*, April 2002.

[46] V. Ramesh, R. L. Glass, and I. Vessey. Research in Computer Science: An Empirical Study. *Journal of Systems and Software*, 70(1–2):165 – 176, 2004.

[47] J. Wainer, C. G. N. Barsottini, D. Lacerda, and L. R. Magalhães de Marco. Empirical evaluation in Computer Science research published by ACM. *Information and Software Technology*, 51(6):1081 – 1085, 2009.

[48] E. M. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Comput. Surv.*, 28(4):626–643, December 1996.

[49] M. V. Zelkowitz and D. R. Wallace. Experimental Models for Validating Technology. *Computer*, 31(5):23–31, May 1998.

[50] W. F. Tichy. Should computer scientists experiment more? *Computer*, 31(5):32–40, May 1998.

[51] T. Schneider, T. Hoefler, and A. Lumsdaine. ORCS: An Oblivious Routing Congestion Simulator. *Indiana University Technical Report, TR-675*, February 2009.

[52] OpenFabrics Alliance. OFED Overview. https://web.archive.org/web/20170405094006/https://www.openfabrics.org/index.php/openfabrics-software.html. [Online; accessed 17-May-2017].

[53] I. McGregor. The Relationship Between Simulation and Emulation. In *Proceedings of the Winter Simulation Conference*, volume 2, pages 1683–1688 vol.2, December 2002.

[54] N. Bunnin and J. Yu. P. In *The Blackwell Dictionary of Western Philosophy*, pages 500–579. Blackwell Publishing, 2007.

[55] J. Elliott. 45 Years of Mainframe Virtualization: CP/67 and VM/370 to z/VM. In *Share Conference in Anaheim*, August 2012.

[56] P. H. Gum. System/370 Extended Architecture: Facilities for Virtual Machines. *IBM Journal of Research and Development*, 27(6):530–544, November 1983.

[57] M. Rosenblum. Vmware's Virtual Platform. In *Proceedings of Hot Chips*, volume 1999, pages 185–196, 1999.

[58] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, G. Regnier, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, and J. Wiegert. Intel® Virtualization Technology for Directed I/O. *Intel® Technology Journal*, 10(3):179—192, 2006.

[59] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIV, pages 205–216, New York, NY, USA, 2009. ACM.

[60] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[61] N. Bobroff, A. Kochut, and K. Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, May 2007.

[62] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. San Francisco, 2008.

[63] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia. A Survey on Virtual Machine Migration and Server Consolidation Frameworks for Cloud Data Centers. *Journal of Network and Computer Applications*, 52:11 – 25, 2015.

[64] J. Shuja, A. Gani, S. Shamshirband, R. W. Ahmad, and K. Bilal. Sustainable Cloud Data Centers: A Survey of Enabling Techniques and Technologies. *Renewable and Sustainable Energy Reviews*, 62:195 – 214, 2016.

[65] A. Shribman and B. Hudzia. Pre-Copy and Post-Copy VM Live Migration for Memory Intensive Applications. In *Euro-Par 2012: Parallel Processing Workshops. Lecture Notes in Computer Science*, pages 539–547. Springer Berlin Heidelberg, 2013.

[66] B. Zhang, X. Wang, R. Lai, L. Yang, Z. Wang, Y. Luo, and X. Li. Evaluating and Optimizing I/O Virtualization in Kernel-based Virtual Machine (KVM). In *IFIP International Conference on Network and Parallel Computing (NPC 2010). Lecture Notes in Computer Science.*, pages 220–231. Springer Berlin Heidelberg, 2010.

[67] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin. Performance Evaluation of Virtualization Technologies for Server Consolidation. *HP Labs Technical Report*, 2007.

[68] S. Crosby and D. Brown. The Virtualization Reality. *Queue*, 4(10):34–41, December 2006.

[69] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal*, 10(3), 2006.

[70] P. Kutch. PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology. *Application note*, 2011.

[71] C. Waldspurger and M. Rosenblum. I/O Virtualization. *Communications of the ACM*, 55(1):66–73, January 2012.

[72] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan. High Performance Network Virtualization with SR-IOV. *Journal of Parallel and Distributed Computing*, 72(11):1471 – 1480, 2012. Communication Architectures for Scalable Systems.

[73] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[74] T. Dillon, C. Wu, and E. Chang. Cloud Computing: Issues and Challenges. In *24th IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33, April 2010.

[75] NRK. Derfor fikk du ikke sjekket selvangivelsen. http://web.archive.org/web/20170517200058/https://www.nrk.no/nordland/altinn-nede-for-telling-1.7560709, 2011. [Online; accessed 17-May-2017].

[76] The Verge. Oslo man's financial records briefly accessible to all after Norwegian tax website crashes. https://web.archive.org/web/20150926083008/http://www.theverge.com/2012/3/22/2892471/norway-altinn-kenneth-tax-leak, 2012. [Online; accessed 17-May-2017].

[77] M. Mao, J. Li, and M. Humphrey. Cloud Auto-scaling with Deadline and Budget Constraints. In *11th IEEE/ACM International Conference on Grid Computing*, pages 41–48, October 2010.

[78] K. S. Rao and P. S. Thilagam. Heuristics Based Server Consolidation with Residual Resource Defragmentation in Cloud Data Centers. *Future Generation Computer Systems*, 50:87 – 98, 2015. Quality of Service in Grid and Cloud 2015.

[79] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes. Toward Energy-Efficient Cloud Computing: Prediction, Consolidation, and Overcommitment. *IEEE Network*, 29(2):56–61, March 2015.

[80] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Sandpiper: Black-box and Gray-box Resource Management for Virtual Machines. *Computer Networks*, 53(17):2923 – 2938, 2009. Virtualized Data Centers.

[81] A. Beloglazov and R. Buyya. Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[82] M. Mishra and A. Sahoo. On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach. In *IEEE 4th International Conference on Cloud Computing*, pages 275–282, July 2011.

[83] A. Corradi, M. Fanelli, and L. Foschini. VM Consolidation: A Real Case Based on OpenStack Cloud. *Future Generation Computer Systems*, 32:118 – 127, 2014. Special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures.

[84] B. Cao, X. Gao, G. Chen, and Y. Jin. NICE: Network-aware VM Consolidation scheme for Energy Conservation in Data Centers. In *20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 166–173, December 2014.

[85] E. Feller, L. Rilling, and C. Morin. Energy-Aware Ant Colony Based Workload Placement in Clouds. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, GRID '11, pages 26–33, Washington, DC, USA, 2011. IEEE Computer Society.

[86] H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya. Virtual Machine Consolidation in Cloud Data Centers Using ACO Metaheuristic. In *Euro-Par 2014 Parallel Processing: 20th International Conference Proceedings*, pages 306–317. Springer International Publishing, Cham, 2014.

[87] A. Beloglazov and R. Buyya. Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers under Quality of Service Constraints. *IEEE Transactions on Parallel and Distributed Systems*, 24(7):1366–1379, July 2013.

[88] H. Zhang, K. Yoshihira, Y. Su, G. Jiang, M. Chen, and X. Wang. iPOEM: A GPS Tool for Integrated Management in Virtualized Data Centers. In *Proceedings of the*

8th ACM International Conference on Autonomic Computing, ICAC '11, pages 41–50, New York, NY, USA, 2011. ACM.

[89] D. Breitgand and A. Epstein. Improving Consolidation of Virtual Machines with Risk-aware Bandwidth Oversubscription in Compute Clouds. In *2012 Proceedings IEEE INFOCOM*, pages 2861–2865, March 2012.

[90] L. Chen and H. Shen. Consolidating Complementary VMs with Spatial/Temporal-Awareness in Cloud Datacenters. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1033–1041, April 2014.

[91] B. Viswanathan, A. Verma, and S. Dutta. CloudMap: Workload-Aware Placement in Private Heterogeneous Clouds. In *2012 IEEE Network Operations and Management Symposium*, pages 9–16, April 2012.

[92] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner. Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 31–40, New York, NY, USA, 2009. ACM.

[93] A. Arcangeli, I. Eidus, and C. Wright. Increasing Memory Density by Using KSM. In *Proceedings of the Linux Symposium*, pages 19–28. Citeseer, 2009.

[94] IEEE. IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 17: Priority-based Flow Control. *IEEE Std 802.1Qbb-2011 (Amendment to IEEE Std 802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011 and IEEE Std 802.1Qbc-2011)*, pages 1–40, September 2011.

[95] V. Jacobson. Congestion Avoidance and Control. *ACM SIGCOMM Computer Communication Review*, 18(4):314–329, August 1988.

[96] G. F. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. pearson education, 2005.

[97] J. Duato, S. Yalamanchili, and L. M. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, 2003.

[98] S. R. Öhring, M. Ibel, S. K. Das, and M. J. Kumar. On Generalized Fat Trees. In *Proceedings of the 9th International Symposium on Parallel Processing*, IPPS '95, pages 37–, Washington, DC, USA, 1995. IEEE Computer Society.

[99] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 77–88, Washington, DC, USA, 2008. IEEE Computer Society.

[100] Y. Ajima, S. Sumimoto, and T. Shimizu. Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers. *Computer*, 42(11):36–40, November 2009.

[101] F. Petrini and M. Vanneschi. k-ary n-trees: High Performance Networks for Massively Parallel Architectures. In *Proceedings 11th International Parallel Processing Symposium*, pages 87–93, April 1997.

[102] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[103] 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*, pages 116–125, September 2008.

[104] E. Zahavi. D-Mod-K Routing Providing Non-Blocking Traffic for Shift Permutations on Real Life Fat Trees. *CCIT Report 776, Technion*, 2010.

[105] 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. Communication Architectures for Scalable Systems.

[106] E. G. Gran. *Congestion Management in Lossless Interconnection Networks*. phd, Faculty of Mathematics and Natural Sciences, University of Oslo, March 2014.

[107] T. Skeie, O. Lysne, and I. Theiss. Layered Shortest Path (LASH) Routing in Irregular System Area Networks. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, pages 8 pp–, April 2002.

[108] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang. Optimized InfiniBand Fat-Tree Routing for Shift All-to-All Communication Patterns. *Concurrency and Computation: Practice and Experience*, 22(2):217–231, 2010.

[109] F. Zahid, E. G. Gran, B. Bogdanski, B. D. Johnsen, and T. Skeie. A Weighted Fat-Tree Routing Algorithm for Efficient Load-Balancing in InfiniBand Enterprise Clusters. In *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 35–42, March 2015.

[110] J. Duato. A Necessary and Sufficient Condition for Deadlock-free Routing in Cut-through and Store-and-forward Networks. *IEEE Transactions on Parallel and Distributed Systems*, 7(8):841–854, August 1996.

[111] B. Zafar, T. M. Pinkston, A. Bermúdez, and J. Duato. Deadlock-Free Dynamic Reconfiguration Over InfiniBand™ Networks. *Parallel Algorithms and Applications*, 19(2-3):127–143, 2004.

[112] R. Pang, T. M. Pinkston, and J. Duato. The Double Scheme: Deadlock-free Dynamic Reconfiguration of Cut-Through Networks. In *Proceedings 2000 International Conference on Parallel Processing*, pages 439–448, 2000.

[113] O. Lysne, J. M. Montañana, T. M. Pinkston, J. Duato, T. Skeie, and J. Flich. Simple Deadlock-Free Dynamic Network Reconfiguration. In *High Performance Computing -*

HiPC 2004: 11th International Conference, pages 504–515. Springer Berlin Heidelberg, 2005.

[114] O. Lysne and J. Duato. Fast Dynamic Reconfiguration in Irregular Networks. In *Proceedings 2000 International Conference on Parallel Processing*, pages 449–458, 2000.

[115] F. O. Sem-Jacobsen and O. Lysne. Topology Agnostic Dynamic Quick Reconfiguration for Large-Scale Interconnection Networks. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, pages 228–235, May 2012.

[116] A. Robles-Gómez, A. Bermúdez, R. Casado, and Å. G. Solheim. Deadlock-Free Dynamic Network Reconfiguration Based on Close Up*/Down* Graphs. In *Euro-Par 2008 – Parallel Processing: 14th International Euro-Par Conference*, pages 940–949. Springer Berlin Heidelberg, 2008.

[117] A. Bermúdez, R. Casado, F. J. Quiles, and J. Duato. Use of Provisional Routes to Speed-up Change Assimilation in InfiniBand Networks. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pages 186–193, April 2004.

[118] W. L. Guay. *Dynamic Reconfiguration in Interconnection Networks*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2014.

[119] J. Liu, W. Huang, B. Abali, and D. K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. In *Proceedings of the USENIX Annual Technical Conference*, pages 29–42, Berkeley, CA, USA, 2006. USENIX Association.

[120] W. L. Guay, S. Reinemo, B. D. Johnsen, C. Yen, T. Skeie, O. Lysne, and O. Tørudbakken. Early experiences with live migration of SR-IOV enabled InfiniBand. *Journal of Parallel and Distributed Computing*, 78:39–52, 2015.

[121] F. Zahid, E. G. Gran, B. Bogdański, B. D. Johnsen, and T. Skeie. Partition-Aware Routing to Improve Network Isolation in InfiniBand Based Multi-tenant Clusters. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 189–198, May 2015.

[122] E. Zahavi, A. Shpiner, O. Rottenstreich, A. Kolodny, and I. 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*, ANCS '16, pages 87–98, New York, NY, USA, 2016. ACM.

[123] A. U. Ranadive. *Virtualized Resource Management in High Performance Fabric Clusters*. PhD thesis, Georgia Institute of Technology, 2015.

[124] Reliable Datagram Socket. http://web.archive.org/web/20170515123339/https://www.kernel.org/doc/Documentation/networking/rds.txt. [Online; accessed 17-May-2017].

[125] OpenFabrics Alliance. Ibnetdiscover Linux Man Page. https://web.archive.org/web/20160702193939/https://linux.die.net/man/8/ibnetdiscover. [Online; accessed 17-May-2017].

[126] H. Rosenstock. Scalable Subnet Administration. In *10th Annual OpenFabrics International Developer Workshop*, 2014.

[127] E. Tasoulas, B. D. Johnsen, and E. G. Gran. System and Method for Providing a Dynamic Cloud with Subnet Administration (SA) Query Caching, October 27, 2015. US Patent Application 14/924,281.

[128] E. Tasoulas, B. D. Johnsen, and E. G. Gran. System and Method for Providing an InfiniBand SR-IOV vSwitch Architecture for a High Performance Cloud Computing Environment, February 23, 2016. US Patent Application 15/050,901.

[129] E. Tasoulas, F. Zahid, B. D. Johnsen, and E. G. Gran. System and Method for Efficient Virtualization In Lossless Interconnection Networks, May 25 2017. US Patent App. 15/210,595.

[130] E. Tasoulas, F. Zahid, B. D. Johnsen, and E. G. Gran. System and Method for Efficient Virtualization In Lossless Interconnection Networks, May 25 2017. US Patent App. 15/210,599.

[131] Efficient and Robust Architecture for the Big Data Cloud (ERAC) Project. https://web.archive.org/web/20170510153854/https://www.simula.no/research/projects/erac-efficient-and-robust-architecture-big-data-cloud. [Online; accessed 17-May-2017].

[132] E. Tasoulas and F. Zahid. ERAC: Efficient and Robust Architecture for the Big Data Cloud. In K. De Bosschere, editor, *ACACES 2014: Poster Abstracts*, pages 209–212. HIPEAC, 2014.

[133] E. Tasoulas, W. L. Guay, S. Reinemo, B. D. Johnsen, C. Yen, T. Skeie, O. Lysne, and O. Torudbakken. Prototyping Live Migration With SR-IOV Supported InfiniBand HCAs. HPC Advisory Council Spain Conference, September 2013.

[134] E. Tasoulas. Using High Performance Network Interconnects in Dynamic Environments. OpenFabrics Alliance (OFA) Workshop, April 2016.

[135] E. Tasoulas, E. G. Gran, B. D. Johnsen, and T. Skeie. A Novel Query Caching Scheme for Dynamic InfiniBand Subnets. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 199–210, May 2015.

[136] E. Tasoulas, E. G. Gran, B. D. Johnsen, K. Begnum, and T. Skeie. Towards the InfiniBand SR-IOV vSwitch Architecture. In *IEEE International Conference on Cluster Computing (CLUSTER), 2015*, pages 371–380, September 2015.

[137] E. Tasoulas, E. G. Gran, T. Skeie, and B. D. Johnsen. Fast Hybrid Network Reconfiguration for Large-Scale Lossless Interconnection Networks. In *IEEE 15th*

International Symposium on Network Computing and Applications (NCA), pages 101–108, October 2016.

[138] F. Zahid, E. G. Gran, B. Bogdański, B. D. Johnsen, T. Skeie, and E. Tasoulas. Compact Network Reconfiguration in Fat-trees. *The Journal of Supercomputing*, 72(12):4438–4467, 2016.