Research Article

SRFabric: A Semi-Reconfigurable Rack Scale Topology

Hu Song ¹, Quan Fang ¹, Yibo Jin ², and Zhuzhong Qian ²

¹Information & Telecommunication Branch, State Grid Jiangsu Electric Power Co. Ltd, Nanjing 210024, China

²State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

Correspondence should be addressed to Zhuzhong Qian; qzz@nju.edu.cn

Received 29 November 2020; Revised 29 March 2021; Accepted 3 April 2021; Published 26 April 2021

Academic Editor: José Francisco Gómez Aguilar

Copyright © 2021 Hu Song et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Rack scale design is a promising trend towards customized hardware design, where high density clusters of SoCs are integrated in the rack. One of the biggest challenges for rack scale computing is the interconnection network. Traditional data center topologies require too many ToR switches to support hundreds of SoCs, while distributed fabrics deliver a considerably high end-to-end latency and network oversubscription. Since no one topology fits all kinds of workloads, a flexible in-rack topology requires a careful redesign to dynamically adapt to diverse data center traffic within tight cost and space constraints in the rack. SRFabric is a semi-reconfigurable rack scale network topology that exploits the high path diversity, the cost-effectiveness of distributed fabrics, and the dynamic reconfigurability of circuit switches. This is accomplished by enabling multiple static ports and dynamic ports for each SoC. Leveraging the partial link reconfigurability, SRFabric is able to optimize its topology to dynamically adapt to various workload patterns. We further propose the design of SRFabric to decide the nearly optimal number of dynamic ports and static ports for expected communication density and performance. Extensive evaluations demonstrate that SRFabric can deliver lower average path length, i.e., 2.21 hops on average, and higher bisection bandwidth, i.e., up to 77% nonblocking bandwidth, and provide comparable performance with state-of-the-art strategy XFabric at a lower cost, i.e., XFabric costs up to 3 times more than that of SRFabric.

1. Introduction

The explosive growth of 2.5 quintillion bytes each day [1] has changed the way a data center is built, managed, and expanded over time. Recent industrial and academic efforts, such as Intel Rack-Scale Architecture [2] proposed, SeaMicro [3], Boston Viridis [4], and Facebook Disaggregated Rack [5], suggest a forthcoming paradigm shift towards rack scale computing, where a rack is mainly composed of hundreds to thousands of interconnected microservers [6, 7], each of which is an energy-efficient and high-density SoC (System on Chip) with a specific unit of CPU, memory, storage, and embedded packet switch. By decoupling these resources, rack scale computing enables finer-grained resource provisioning and independent upgrading of CPU, memory, and storage, leading to a lower capital and operational cost.

Essentially, the rack scale computing subsequently affects the in-rack network, since a higher density of SoCs

within a rack imposes serious constraints on switching cost and power consumption. For example, interconnecting 512 SoCs, each with 100 Gbps of bandwidth, requires a ToR (Top of Rack) switch with 512 ports and 51.2 Tbps of bisection bandwidth. However, such a high-radix and high-bandwidth switch is very expensive and power consuming. Except for those expensive ToR switches, distributed fabrics (e.g., Mesh [8] and Torus [9]) are widely studied, which are more cost-effective through forwarding packets by SoCs themselves. However, forwarding packets through multiple hops incur relatively high end-to-end latency and network oversubscription, significantly degrading the performance of latency-sensitive applications. As a result, a static topology is hard to adapt to a variety of workloads in multitenants data centers, which is challenging in the design of in-rack network. That is, one-topology-fits-all topology design is nonexistent.

To address these challenges, some researchers [10–13] have explored the reconfigurable topology and improved the

network oversubscription through reconfiguring the connectivity of ToR switches. Unfortunately, these datacenter-scale designs are not suitable for in-rack interconnections since they often require plenty of expensive high-radix ToRs. In contrast, works such as XFabric [14, 15] maintain the benefits of distributed fabrics, while they enable the links to be dynamically reconfigured by attaching each port of a SoC to the crossbar circuit switch. However, aggressively providing full link reconfigurability fails to earn performance-cost benefits, since different traffics have various performance sensitivities to the reconfigurability. That is, we should take the trade-off between the degree of reconfigurability and the cost, with only a slightly tolerable performance penalty.

Motivated by this intuition, this paper presents SRFabric (a semi-reconfigurable rack scale fabric) that takes both cost and path diversity into consideration through reconfiguring the ports. To achieve this, the ports of a SoC are partitioned into two sets: the static part and the dynamic part. Each static port of a SoC is connected to the corresponding static port of its neighbour, while each dynamic port of a SoC is attached to the circuit switch. Note that the dynamic circuit link between each pair of SoCs is reconfigured and established by connecting the corresponding dynamic ports on the circuit switch. A novel topology algorithm needs to be explored via determining the configuration of dynamic ports in the circuit switch over time, in order to optimize the network throughput as well as the cost. SRFabric actually enables billions of possible topologies in response to various traffic demands and shows unprecedented flexibility at a reasonable cost.

SRFabric differs from other works such as XFabric since it sacrifices full link reconfigurability for partial link reconfigurability at a lower cost. Although partial link reconfigurability limits the possibility regarding various topologies, an acceptable expense for commercial rack sizes and desired communication densities is much more important for commercial deployment. Therefore, our objective is to minimize the overall cost of ports, while still achieving performance guarantee for expected communication densities.

Nonetheless, it is not trivial to determine in advance the number of dynamic ports and static ports, due to an exponential number of potential workload patterns and allocation schemes. As shown in our case studies later, the Collect Pattern (i.e., 1-to-all traffic [16, 17]) performs the worst in terms of the reconfigurability and requires plenty of dynamic ports to ensure the desired level of traffic performance. Based on this observation, we first analyze the distribution in terms of the shortest path length for the Collect Pattern and further generalize the design of the topology for desired communication density by studying the relationship between the density of links, the density of nodes, i.e., SoCs, and the reconfigurability. We then detail the design of our proposed algorithm SRFabric that searches the nearly optimal number of dynamic ports and static ports effectively and efficiently.

At last, we evaluate the performance of our proposed algorithm SRFabric upon realistic traffic patterns by

comparing it with several state-of-the-art strategies, including 3D Torus and XFabric. The results show that SRFabric outperforms 3D Torus with higher bisection bandwidth and lower average path length, incurring only 2.21 hops on average, and delivers almost similar performance to XFabric at a 29% reduction on the cost. Besides, these evaluations demonstrate that a higher level of reconfigurability cannot translate into salient benefits when the communication density is high.

2. SRFabric Architecture

In the section, we present the design of SRFabric, i.e., it uses the partial link reconfigurability at a reasonable cost. The summary of all main notations used is illustrated in Table 1.

2.1. Overview of SRFabric. SRFabric is a partial reconfigurable rack scale network topology that combines dynamic circuit switch with static distributed fabrics. The rationale behind SRFabric is that an appropriate degree of reconfigurability can cost-effectively ease the inherent flaw of long path length in distributed fabrics. Specifically, a SRFabric rack consists of n SoCs, each of which exposes s static ports and d dynamic ports in an embedded packet switch. Each static port of a SoC is connected to one static port of its neighbour, while each dynamic port of a SoC is connected to one port of the circuit switch. Besides, each circuit switch handles inter-rack traffic by l uplinks that are determined by the expected use of a SRFabric rack. Thus, SRFabric has $d * l$ uplinks in total. For convenience, we denote a n -sized SRFabric topology with s static ports, d dynamic ports, and n SoCs as an instance, i.e., SRFabric (n, s, d, l). We defer the discussion of how s and d should be optimized for different rack sizes and communication densities in Section 3.

Figure 1 shows a simple instance: SRFabric (12, 4, 2, 2), where all the static links form a multihop direct-connect 2D Torus topology. Each SoC is able to communicate with any other SoCs via multiple hops in static distributed fabric. Although the design of interconnections for static ports is not necessarily restricted to Torus, it actually offers the benefits over other distributed fabrics (e.g., Mesh) such as lower average path length and higher throughput. Nonetheless, frequent “eastern-western” traffic (e.g., Web Search) will inevitably experience a high count of hops and high latency in Torus, leading to a violation of QoS (Quality of Service).

Remark 1. We should mention here that all static links form a multihop direct-neighbours connect 2D Torus topology. 2D Torus is two dimension with degree of four for each node. And, each node connects its four nearest and corresponding nodes on its opposite edges in a rectangle. The connection of opposite edges can be visualized by rolling the rectangular array into a “tube” and then bending the “tube” into a torus to connect the other two [18]. Such connections ensure that each node directly connects its nearest neighbours, and its furthest neighbours if it is located on the edge

TABLE 1: Summary of main notations in this paper.

Symbol	Description
(n, s, d, l)	SRFabric parameters: a Rack with n SoCs, s static, d dynamic ports, and $d \cdot l$ uplinks
c_s, c_d	Costs of static link and dynamic port, respectively
$T_{\text{node}}, T_{\text{link}}$	Node density and link density, respectively
$F_n^{s,d}(k)$	Probability that SPL ¹ is less than k hops
P_{target}	Desired probability in terms of SPL for a topology
$\xi_n(k)$	Probability that the distance on n SoCs exceeds k

¹denotes the term ‘‘SPL’’ which means the shortest path length.

FIGURE 1: SRFabric architecture with $n = 12$, $s = 4$, $d = 2$, and $l = 2$. Red lines are used for dynamic links, while black lines are used for static links.

of the rectangle. Furthermore, the maximal hops between node pairs are also ensured, i.e., the worst performance using just static links. We should also mention here that efficient and scalable routing algorithms are proposed for 2D Torus [19].

To overcome the inherent defect of static distributed fabrics, we resort to dynamically establishing a dedicated circuit link between the source SoC and the destination one through the circuit switch. More specifically, SRFabric has d independent circuit switches with n SoC ports and l uplink ports. We assume that the values of n and l are both even for simplicity. Available commercial circuit switch supports approximately 350 ports [14]. By using 350 ports, a full folded Clos network architecture could be established as an alternate substitute of a single large circuit switch, which involves multiple smaller circuit switches for communications. Each dynamic port of a SoC is attached to the ports of d circuit switches, where direct circuit links are dynamically reconfigured in response to heavy traffic demands. Since uplink ports are scarce and costly, there is no need to directly connect each port of a SoC to one of the uplink port of the circuit switch. Instead, the SoC could forward its traffic by using a bridge SoC if needed, which has already been connected to an uplink port.

Furthermore, SRFabric is able to simultaneously provide multiple dedicated high-capacity links, especially when the system requires much higher bandwidth than just the capacity of a single link. For example, when SoC A wants to communicate with SoC B, requiring α ($\alpha \leq d$) times the bandwidth of a single link, SRFabric will set up α A-B circuit links over the corresponding α circuit switches. Note that the maximum available bandwidth is determined by d . The appropriate reconfiguration is decided and executed in real time by a central manager. By dynamically adjusting the bipartite mapping of d circuit switches, SRFabric improves the delay of communications and alleviates the bandwidth bottlenecks effectively in-rack scale.

2.2. Reconfiguration Mechanism. Each SRFabric rack has a lightweight controller that monitors the network load in the rack and manages the reconfiguration in response to various traffic demands, such as those SDN controllers [20, 21]. The controller has a global visibility into the state of the SoCs and circuit switches. Concretely, each SoC maintains a demand vector and records the traffic sent to each SoCs within the rack. At regular interval, this traffic information is actively sent to the central controller and aggregated as a global traffic demand matrix. The controller optimizes and then specifies the port mapping for each circuit switch such that the network throughput is maximized.

Intuitively, it is observed that a shorter routing path, especially for SoC pairs with high traffic demand, occupies less link resources, improving the network utilization effectively. We therefore use a greedy algorithm that establishes dynamic circuit link preferentially for SoC pairs with highest demands. To do so, the controller sorts all the SoC pairs in descending order according to their traffic demands and iteratively permits the SoC pairs with the highest traffic over circuit link until there is no vacant dynamic port on the circuit switches. If there are multiple circuit switch candidates satisfying the traffic demand of a SoC pair, SRFabric further selects one of them to establish the circuit link on the consideration of side effect. Note that different links established by these candidates have quite different side effects to other SoCs, i.e., the forwarded traffic incurred on these bridges.

Once the circuit configuration is determined, the controller proceeds to record the physical topology and compute the shortest path routing for each pair of SoCs by BFS (Breadth First Search). Note that the shortest path routing can be seamlessly replaced with any other standard routing

schemes in order to satisfy various application requirements. If multiple shortest path routing schemes are available for a pair of SoC at the same time, we rank the schemes according to the degree of path congestion and choose the one with the least congested path. The outputs are a set of routing tables that are used to control the data plane of the SRFabric rack.

2.3. System Model. We then illustrate the system model of SRFabric in terms of the cost and our proposed problem.

The cost in the objective of our proposed problem contains two parts: the static links and the dynamic ports. In practice, static links are directly printed on PCB board, incurring a little in terms of the cost compared with the dynamic ports on the circuit switch. Therefore, the cost of SRFabric mainly depends on the cost of partial link reconfigurability on the circuit switch. Assume that the costs of static links and dynamic ports are c_s and c_d , respectively. The cost of SRFabric is in proportion to the rack size n and is defined as

$$\text{Cost} = n * \left(d * c_d + s * \frac{c_s}{2} \right). \quad (1)$$

Compared with XFabric, SRFabric sacrifices full link reconfigurability for partial link reconfigurability at an acceptable cost. To make a fair comparison with XFabric, as shown in Table 2, the per-port cost is 3\$ for a commercial crossbar circuit switch and the per-link cost is 0.2\$ for static links according to previous works [14]. SRFabric also behaves better than XFabric regarding the cost at different SoC densities. For example, when the number of SoC is 256, both SRFabric and XFabric use a single crossbar circuit switch with 256 ports. The overall ports needed for SRFabric (256, 4, 2, 2) are approximately 516, while XFabric (256) requires 1792 ports in total. In terms of the cost, XFabric (256) requires 3.4 times much more than that of SRFabric (256, 4, 2, 2), i.e., 1650.4\$.

To evaluate the performance of SRFabric, we consider the most important and commonly used metric: the shortest path length (SPL) when the network topology is optimized in terms of the efficiency. Obviously, a shorter SPL will reduce the message latency and increase the network utilization. Here, $\text{SPL}(i, j)$ refers to the shortest path between SoC i and j . Although the value of $\text{SPL}(i, j)$ is unchanged in terms of the static links, SRFabric can reconfigure the dynamic links and shorten the shortest path length effectively. Therefore, it is necessary to study the distribution of SPL between a random pair of SoCs i and j over all the possible topologies configured. Note that SRFabric only considers the value of SPL instead of the relative positions among SoCs.

Figure 2 illustrates the SPL distributions of both SRFabric and XFabric with 512 SoCs. We fix the number of static ports to 4 and change the number of dynamic ports from 1 to 5 for SRFabric. It is observed that SRFabric ($d=1$) behaves the worst due to the highest value of average SPL while SRFabric ($d=5$) performs the best. The probability of $\text{SPL} \leq 3$ increases by using SRFabric with the growth of the number of dynamic ports. Specifically, the probability of $\text{SPL} \leq 3$ is up to 75.69% at $d=3$, and even reaches 93.1% at

TABLE 2: Preliminary case study for typical crossbar circuit switch [14].

Content	Dynamic ports	Static links	Expense (\$)
Per port	1	0	3
Per link	0	1	0.2
XFabric (256)	1792	0	5376
SRFabric (256, 4, 2, 2)	516	512	1650.4

FIGURE 2: SPL distributions of XFabric and SRFabric, respectively.

$d=5$. However, the improvement of SPL results in a marked increase in terms of the circuit switch cost.

Besides, SRFabric ($d=2$) shows a similar SPL distribution compared with XFabric. That is, SRFabric ($d=2$) and XFabric achieve 60.8% and 60.4%, respectively, regarding the probability of $\text{SPL} \leq 3$. Particularly, almost 97% of SoC pairs have a shortest path length less than 4 hops for both SRFabric ($d=2$) and XFabric. The reason behind is that the reconfigurability earns few performance improvement for all-to-all communication. In real data centers, most realistic workloads have lower but different communication densities than that of all-to-all pattern. It is difficult to decide the optimal reconfigurability scheme at a reasonable cost for a variety of workloads, since partial link reconfigurability limits the topologies and actually impacts the performance. Fortunately, IT designers have a rich knowledge of the expected workload sets and relevant communication densities. It is possible to customize the number of both static and dynamic ports for expected communication performance at a reasonable cost for realistic usage.

In the following, we propose the design of SRFabric for expected rack use in Section 3 and further show that SRFabric ($d=2$) provides comparable performance for common workloads at only 29% cost of XFabric in Section 4.

3. Design for Expected Communication

Intuitively, with the growth of the number of dynamic ports, the topology space considered by SRFabric also increases. As

a result, SRFabric has higher demands regarding both computation and storage, leading to a higher cost. Besides, the performance in terms of the traffic between SoC pairs is also influenced by the growth number of dynamic ports. Therefore, it is necessary to carefully treat the tradeoff between the cost, the reconfigurability, and the performance issues in order to optimize the overall topology design for different rack sizes and communication densities.

In this section, we optimize both the number of static ports and the number of dynamic ports for different rack sizes and communication densities. We first start with a problem definition, followed by a detailed algorithm description.

3.1. Problem Formulation. Without loss of generality, we use the communication density to capture a set of workload patterns given the number of nodes (SoCs) and links, including both node density and link density. We use T_{node} and T_{link} to denote the densities of nodes and links, respectively. Here, T_{node} counts the number of SoCs having communications with others and T_{link} counts the number of SoC pairs having the communications. Note that (T_{node}/n) refers to the ratio of nodes having communications. Thus, T_{node} can be seen as the node density; Similarly, (T_{link}/n^2) refers to the ratio of links having communications. Thus, T_{link} can be regarded as the link density. Since each node has at least one link, we have

$$\frac{T_{\text{node}}}{2} \leq T_{\text{link}} \leq T_{\text{node}} \frac{(T_{\text{node}} - 1)}{2}. \quad (2)$$

Note that the possible number of workload patterns increases with the growth of both T_{node} and T_{link} . In this paper, we assume that the source nodes and the destination nodes for each workload are given before the decisions regarding both static and dynamic ports, since the communications can be only incurred after the deployment of applications. And, after the deployment of each component of an application, its workload patterns are often determined, e.g., the NameNode in HDFS [22] manages the meta information for stored data blocks, and it needs to communicate with all DataNodes in the system. In contrast, the ApplicationMaster used in MapReduce [23] for each job, i.e., the controller of a job, only communicates with its own running tasks.

Given the expected communication density, we may allocate more dynamic ports per SoC to pursue a lower value of SPL. However, the cost regarding the ports should be controlled. Therefore, we should optimize the dynamic ports and static ports for expected communication density in order to take an effective tradeoff between the expected performance and the incurred cost. In order to describe the desired performance, i.e., the SPL of any SoC pair in the rack is less than k hops, we use $F_n^{s,d}(T_{\text{node}}, T_{\text{link}}, k)$ to represent the probability that the SPL is controlled less than k hops under the worst workload pattern. We also use P_{target} to represent the desired performance for given communication density $(T_{\text{node}}, T_{\text{link}})$. By applying the cost defined for

SRFabric in equation (1), we formulate the topology design optimization problem for SRFabric as follows:

$$\begin{aligned} & \text{minimize Cost} \\ & \text{subject to } F_n^{s,d}(T_{\text{node}}, T_{\text{link}}, k) \geq P_{\text{target}}. \end{aligned} \quad (3)$$

The objective is to minimize the incurred cost while ensuring the desired performance for expected communication density. Although the domains of both s and d are small, $F_n^{s,d}(T_{\text{node}}, T_{\text{link}}, k)$ is still hard to be solved precisely because of an exponential number of potential workload patterns. It is more challenging that there are multiple possible schemes for SoC deployment and plenty of corresponding schemes for switch reconfiguration given various workload patterns. Since the strategy of greedy search is too heavy-lifting, we instead resort to a theoretical analysis regarding the SPL distribution first for desired communication density.

Intuitively, if one SoC wants to communicate with many peers, more dynamic ports are required to provide similar performance guarantee in the worst deployment. Unfortunately, the number of dynamic ports is limited. For example, the pattern of 1–100 requires up to 100 dynamic ports to ensure 1-hop SPL, while a ring pattern of 100 SoCs only needs 2 dynamic ports per SoC. Therefore, the requirement of Collect Pattern regarding both static and dynamic ports, i.e., 1 to all mode, refers to the worst case among all scenarios with the same node density and link density. In the following, we first start with the worst case, i.e., the Collect Pattern in Section 3.2, and then generalize the discussion to any communication density in Section 3.3.

3.2. Worst Case: The Collect Pattern. To analyze the SPL distribution of the Collect Pattern, we first need to know the SPL distribution of a random SoC pair in SRFabric. Assume that a random pair of SoC i and j is directly connected with probability p . Considering that i and j could be directly connected through either static links or d crossbar circuit switch, p is expressed as

$$p = 1 - \left(1 - \frac{s}{n-1}\right) \left(1 - \frac{1}{n-1}\right)^d, \quad (4)$$

where the product term refers to the probability of unsuccessful connection. For static links, the connection probability is $s/(n-1)$, while for dynamic links, the connection probability for each one is $1/(n-1)$. Note that there are d dynamic links.

Therefore, SRFabric can be seen as a random regular graph with a direct connection probability p . However, it is still very hard to precisely estimate the SPL distribution regarding a random regular graph [24]. In this paper, we thus analyze the SPL distribution approximately by constructing a SPL tree. Concretely, one SoC i is randomly picked as the root node, and the rest SoCs are located at the corresponding levels according to the paths from the root SoC. The number of SoCs is denoted by $\bar{N}_k$, which records the shortest path distance $\text{SPL} > k$, $k = 0, 1, 2, \dots, n-1$, from the root node. Similarly, we denote the number of SoCs

exactly at a shortest path distance $SPL = k$ by N_k , where $N_0 = 1$ and $N_k = \overline{N}_{k-1} - \overline{N}_k$. Assume that the tail distribution $\xi_N(k)$ of SPL is the probability that the shortest path length between a random pair of SoCs is larger than k , and we have [25]

$$\xi_n(k) = \frac{\overline{N}_k}{\{n-1\}}. \quad (5)$$

Clearly, the tail distribution $\xi_n(k)$ decreases as the number of dynamic ports d grows. By further analyzing the SPL tree, the following theorem, i.e., Theorem 1, is ensured.

Theorem 1. *For a random pair of SoCs in SRFabric, the tail distribution of the shortest path length obeys*

$$\xi_n(k+1) = \xi_n(1)(1-p)^{(n-1) \cdot (\xi_n(0) - \xi_n(k))}. \quad (6)$$

Proof. Considering that each SoC at a shortest path distance $SPL = k+1$ has at least a direct connection with SoCs at a shortest path distance $SPL = k$, it could be derived that N_k obeys the following recursive equation [26]:

$$N_{k+1} = \overline{N}_k(1 - (1-p)^{N_k}). \quad (7)$$

By replacing N_{k+1} with $\overline{N}_k - \overline{N}_{k+1}$, it equals to

$$\overline{N}_{k+1} = \overline{N}_k(1 - p)^{\overline{N}_{k-1} - \overline{N}_k}, \quad (8)$$

where $\overline{N}_0 = n-1$ and $\overline{N}_1 = (n-1)(1-p)$. According to the recursion relationship illustrated in equation (8), we can estimate the number of SoCs at a distance $SPL = k$ and further get the distribution of the shortest path length from the root to the other $n-1$ SoCs. Since the root SoC is randomly picked, the relationship between it and the other SoCs equals to a random pair of SoCs regarding the SPL.

Therefore, combining equations (8) and (5) yields the second-order difference relationship of ξ_n :

$$\xi_n(k+1) = \xi_n(k)(1-p)^{(n-1)(\xi_n(k-1) - \xi_n(k))}, \quad (9)$$

where $\xi_n(0) = 1$ and $\xi_n(1) = 1-p$. By making an iteration on k , it can be further simplified as equation (6).

Theorem 1 also provides a worst case analysis regarding SPL distribution for all-to-all pattern. For example, it can be written as $\xi_n(4)$ equals to 9.72% at $n = 512$ and $d = 2$. This ensures that more than 90% SoC pairs have a shortest path length less than 4 hops in SRFabric ($d = 2$). In this case, reconfigurability has little to improve the performance, since there are no free circuit links for all-to-all pattern, i.e., each node obeys the worst case, the Collect Pattern.

In fact, the communication density limits the benefit of reconfigurability, since partial link reconfigurability in SRFabric restricts the exchange between any two SoCs in the SPL tree. As the link density increases, more dynamic ports on each SoC are required to maintain the same performance, leading to a severe pressure on the cost and other resource requirements within a rack. In other words, both communication density and dynamic ports determine

the performance. For example, $d = 2$ in SRFabric means that each SoC is able to change two neighbour SoCs at most. If more than d SoCs have communications with the source SoC, the shortest paths between those non-neighbour SoCs regarding the source and the source itself must require the help from the neighbours. Fortunately, each SoC can also change its d neighbours and shorten the hops to facilitate the communication for a pair of SoCs. To further understand the behavior of the reconfigurability, we denote the maximal number of reconfigurable SoCs in the k level of the SPL tree by R_k , where $R_0 = 0$ and $R_1 = d$. If the value of R_k is large, a lot of efforts are required for SoC reconfigurability, since the switch has considered the worst case. Generally, it is expected that we should put as many SoCs as possible in the first four levels of the SPL tree through reconfiguration and exchanging of the SoC positions for any possible traffic patterns and deployment. $\square$

Theorem 2. *SRFabric ensures $(1 - \xi_n(1) - \xi_n(3) + \xi_n(2)) * d$ percentile of SoCs in the SPL tree, which have a shortest path length of $SPL \leq 4$ in the worst deployment.*

Proof. Intuitively, in the worst allocation, each SoC can be located in the level $k \leq 4$ if and only if at least one dynamic circuit link is used for tracing the shortest path from the source. Therefore, we only need to estimate the maximal number of potential dynamic links in the first four levels of the SPL tree, which is denoted as $\overline{R}_{k \leq 4}$.

Since each SoC has d dynamic ports, we have $R_1 = d$ in the level $k = 1$ of the SPL tree. In the level $k = 2$ of the SPL tree, each of the $N_1 - R_1$ SoCs in the first level can occupy d dynamic links at most while each of the R_1 SoCs can only occupy $d-1$ dynamic links. Note that one dynamic port of these SoCs has been used to connect to the SoCs in the upper level. Through a recursive way, R_k can be expressed as

$$R_k = N_{k-1} * d - R_{k-1}. \quad (10)$$

Considering the first four items of R_k , the number of possible dynamic links is expressed as $\overline{R}_{k \leq 4} = (N_1 + N_3) * d$. Replacing N_1 and N_3 by using equation (5), we further have

$$\overline{R}_{k \leq 4} = (1 - \xi_n(1) - \xi_n(3) + \xi_n(2)) * d * (n-1). \quad (11)$$

Equation (11) implies that $(1 - \xi_n(1) - \xi_n(3) + \xi_n(2)) * d$ percentile of SoCs in the worst case is able to lie in the first four levels of the SPL tree, and we complete the proof.

Theorem 2 actually provides a theoretical explanation on how the reconfigurability and the communication density impact the performance. Suppose that $d = 2$ and $n = 512$, it can be derived that 63.64% SoCs have a shortest path length of at most 4 hops in the worst case. Based on these two theorems, we can further infer how the SPL distribution of the Collect Pattern changes as node density increases.

In the Collect Pattern, we have $T_{\text{link}} = T_{\text{node}} - 1$. As T_{node} increases, more SoCs have to suffer from a higher SPL. In the

Input: $T_{\text{node}}, T_{\text{link}}$: node density and link density, respectively; P_{target} : desired probability of SPL.
Output: s, d : static ports and dynamic ports, respectively.

- (1) Initialize $s = 2$ and $d = 1$;
- (2) while true do
- (3) $h \leftarrow$ The deepest level that contains T_{node} nodes;
- (4) if $T_{\text{link}} \leq (T_{\text{node}}/2) + d - 1$ then
- (5) All the connections are within 1 hops;
- (6) else if $(T_{\text{node}}/2) + d - 1 < T_{\text{link}} \leq T_{\text{node}} - 1$ then
- (7) $T_{\text{node}} - T_{\text{link}} - 1$ connections have 1-hop SPL;
- (8) Others $\leftarrow$ collect $(2T_{\text{link}} + 2 - T_{\text{node}})$;
- (9) else if $T_{\text{node}} - 1 < T_{\text{link}} \leq \sum_{i=1}^d \{T_{\text{node}} - i\}$ then
- (10) $\sum_{i=1}^{h-1} \{T_{\text{node}} - i\}$ connections $\leftarrow$ collect (T_{node}) ;
- (11) Others $\leftarrow$ collect $(T_{\text{link}} + 1 - \sum_{i=1}^{h-1} \{T_{\text{node}} - i\})$;
- (12) else
- (13) $\sum_{i=1}^{h-1} \{T_{\text{node}} - i\}$ connections $\leftarrow$ collect (T_{node}) ;
- (14) Other connections are within $2h$ hops;
- (15) end if
- (16) Compute $F_n^{s,d}(T_{\text{node}}, T_{\text{link}}, k)$;
- (17) if $F_n^{s,d}(T_{\text{node}}, T_{\text{link}}, k) < P_{\text{target}}$ then
- (18) Change s, d by using gradient descent method;
- (19) else
- (20) return s, d ;
- (21) end if
- (22) end while

ALGORITHM 1: SRFabric design algorithm.

extreme case that one SoC communicates with all the other SoCs, reconfigurability shows a weak promotion on the performance. Assume that all the SoCs are randomly deployed; then, all the T_{node} SoCs need to bypass dynamic ports to communicate with the source in the reconfiguration. According to Theorem 2, the worst allocation allows at most R_k SoCs on the level k of the SPL tree until all the T_{node} SoCs have been placed. Therefore, we can divide the value of T_{node} into several intervals by R_k and obtain the SPL distribution of Collect Pattern. For example, suppose $d = 2$ and $n = 512$, and it can be derived that $R_1 = 2, R_2 = 10, R_3 = 58$, and $R_4 = 254$. That means SRFabric can guarantee a shortest path of 4 hops for $T_{\text{node}} \leq 324$ and 3 hops for $T_{\text{node}} \leq 70$, no matter where the positions and the nodes of the Collect Pattern are allocated. Furthermore, we generalize the SPL distribution of Collect Pattern to any communication density and propose the design of SRFabric (Algorithm 1). $\square$

3.3. General Case: Desired Density. Following the basic intuition of the reconfigurability, the communications always try to use the dynamic ports to shorten the shortest path length as possible. Therefore, we can construct a worst traffic pattern for each desired communication density, in which each shortest path uses one dynamic port at least. Note that the worst deployment is needed for ensured traffic performance. As we mentioned before, the worst case is to simply allocate as many connections as possible to one random SoC and then repeat the allocations until all the connections are used. Intuitively, it can be seen as a group of interconnected Collect Patterns. Based on the discussion of the worst traffic

pattern, we compute the probability $F_n^{s,d}(T_{\text{node}}, T_{\text{link}}, k)$ for any desired communication density and related required dynamic ports.

Algorithm 1 finds the nearly optimal number of dynamic ports and static ports, which guarantees the performance of any possible traffic patterns with given link density and node density. collect(T_{node}) refers to the deployment conducted by the Collect Pattern with T_{node} nodes based on the SPL distribution. The main idea is to tackle the SPL distribution for expected communication density according to the Collect Pattern until the performance is guaranteed for the worst workload pattern. Formally, we exploit the dynamic ports to shorten the SPL of these T_{link} connections and then analyze the interrelationships between T_{link}, d , and T_{node} . T_{link} is thus divided into four intervals as follows:

- (1) $T_{\text{link}} \leq (T_{\text{node}}/2) + d - 1$: in this case, any requirement of 1-hop circuit link can be established for any connection, since each node has delivered one connection at least.
- (2) $(T_{\text{node}}/2) + d - 1 < T_{\text{link}} \leq T_{\text{node}} - 1$: in this case, the worst case behaves as the Collect Pattern does. Intuitively, $T_{\text{node}} - T_{\text{link}} - 1$ connections have the shortest paths of 1 hop, while the other $2T_{\text{link}} + 1 - T_{\text{node}}$ connections show the same SPL distribution as collect $(2T_{\text{link}} + 1 - T_{\text{node}})$.
- (3) $T_{\text{node}} - 1 < T_{\text{link}} \leq \sum_{i=1}^d \{T_{\text{node}} - i\}$: in this case, for each SoC, we greedily connect it to all of the others and then repeat the link allocations until no connections are available. Therefore, the worst pattern is the combination of at most d trials of Collect Patterns. According to Theorem 2, $\sum_{i=1}^{h-1} \{T_{\text{node}} - i\}$

connections follow the SPL distribution of $\text{collect}(T_{\text{node}})$, while the other connections follow the SPL distribution of $\leftarrow \text{collect}(T_{\text{link}} + 1 - \sum_{i=1}^{h-1} \{T_{\text{node}} - i\})$. Note that h is the deepest level of SPL tree, which is updated during the “while” loop regarding the SPL distribution.

- (4) $\sum_{i=1}^d \{T_{\text{node}} - i\} < T_{\text{link}}$: intuitively, the fact always holds that $\sum_{i=1}^{h-1} \{T_{\text{node}} - i\}$ connections follow the distribution of $\text{collect}(T_{\text{node}})$. However, it is difficult to reveal the SPL distribution for additional connections. Considering that there always exists a path of at most $2h$ hops for additional connections using the root of the SPL tree as the bridge, the average SPL of all additional connections is nearly 2 hops.

Once the SPL distribution for the expected communication density is solved, we validate whether the desired probability P_{target} is satisfied. If not, we change s and d in the most cost-effective way by computing the corresponding gradient and continue the search process until the design is determined. Although SRFabric sacrifices full link reconfigurability for partial link reconfigurability at an acceptable cost, the SRFabric design provides satisfactory performance guarantee as expected. The cost incurred by Algorithm 1 is acceptable, which is measured in the experiments.

Remark 2. In this paper, we focus on the decisions for rack-scale topology given a batch of workload patterns. Although our proposed problem and algorithm are designed for an offline setting, such formulation and schema can be directly used for dynamic deployment over time if the algorithm is frequently triggered. The trigger conditions could be as follows: (1) the maximal hops regarding the communications between SoC pairs incurred by the latest submitted applications are too long, (2) the number of latest submitted applications exceeds a pre-given threshold, and (3) the workload patterns of latest submitted applications are quite different from the existing ones. In any case, as long as the decisions need to be updated or made, the formulation and algorithm are unchanged.

Remark 3. Traditionally, each link has limited bandwidth capacity, e.g., the capacity of a single link could be hundreds and thousands of Mbps (Megabit per second) or more [13, 14] between SoCs and the switch, which is often adequate for applications within a rack. Although the bandwidth requirements between SoCs are quite different [22, 23, 27], if the bandwidth requirements between a fixed pair of SoCs do not exceed the related link capacity, they can be regarded as a single communication, i.e., those communications can be aggregated as a single logical communication. Furthermore, if the bandwidth requirements between a given pair of SoCs exceed the capacity of a single link, multiple links hosting multiple logical communications are then needed in order to ensure the performance. Note that each logical communication may contain multiple real communications as long as the requirements of those real communications with the same source and destination do not exceed the link capacity. Then, the aggregated

communications, i.e., the connectivity of those logical communications with the same bandwidth requirement, are actually considered as the input.

4. Evaluation

In our evaluation, we answer the following four questions. (i) Compared to static topologies and reconfigurable topologies, is SRFabric effective enough in achieving high bisection bandwidth, low path length, and high path diversity across a variety of workloads? (ii) How does the number of dynamic ports significantly impact the performance of SRFabric? (iii) How does the communication density impact the benefit of reconfigurability? (iv) Is the proposed SRFabric algorithm efficient enough?

4.1. Methodology

4.1.1. Metrics. The main metrics we focusing on in trace-driven simulations contain three parts: the average path length, the bisection bandwidth, and the cost for reconfiguration. The average path length reflects the end-to-end latency, while the bisection bandwidth reflects the transmission capacity. Meanwhile, the cost for reconfiguration dominates the design since an acceptable cost is suitable for commercial use.

4.1.2. Algorithms. To evaluate the performance from different dimensions, we compare SRFabric against commonly used 3D Torus and the state-of-the-art strategy XFabric. 3D Torus is a static distributed topology with high path length and low bisection bandwidth, while XFabric is proposed in-rack scale computing with full link reconfigurability. Unless otherwise stated, we run the workloads as the instance of SRFabric (512, 4, 2, 2). 512 is also an appropriate size to form a 3D Torus topology where each dimension is 8. Our intention is to provide an intuitive illustration of the borderline performance. For XFabric, we assume there are six internal circuit switches with 512 ports and one uplink circuit switch with 516 ports.

4.1.3. Workloads. The workload patterns are derived from diverse real-world traces and synthetic traffic:

“Map-Reduce” is the most commonly used workload in data centers. In this paper, we use a publicly available cluster trace [28], which is sampled randomly from the original trace. Each host in the trace is mapped onto the SoCs randomly without any topology-aware workload placement.

“Graph Processing” is also a widely used workload pattern, for feeding the clustering techniques and the detection of cliques in social network. For XFabric, we use the LiveJournal trace and divide the graph into multiple uniform partitions by using METIS, such that the connections between these partitions are low. Each partition is mapped to one SoC and the traffic is proportional to the connections.

“Random” is a synthetical traffic pattern that is used to quantify the effect of the topology with high communication density on performance. For each evaluation, each SoC

randomly picks different numbers of destination SoCs from 6 to 512 and sends traffic in a constant bit rate.

4.2. Evaluation Results

4.2.1. Average Path Length. We first explore the performance of SRFabric regarding the latency, which is intuitively indicated by the average path length. Figure 3(a) illustrates the average shortest path length of each flow over various traffic patterns. It is observed that 3D Torus achieves the largest average path length, i.e., at least 5.2 hops, under all workloads and provides a moderate performance. SRFabric ($d=1$) achieves an average path length of 3.23 hops for Map-Reduce workload and 3.84 hops for graph processing workload. When adding one more dynamic port for each SoC, SRFabric ($d=2$) reaches an average path length of 2.21 hops for Map-Reduce and 2.5 hops for graph processing. In contrast, XFabric performs well, with an average path length of 1.8 hops and 2.41 hops, respectively. These results suggest that SRFabric ($d=2$) achieves comparable performance with XFabric, which not only is better than SRFabric ($d=1$) but also costs only 29% of XFabric. Besides, compared with 3D Torus, both SRFabric and XFabric show minor improvement on average path length for random workload, nearly 5 hops. The performance gap is derived from the higher density of source-destination flows among all possible SoC pairs in random workload. It implies that the reconfigurability is able to improve the performance of applications, but is limited by the workload patterns.

4.2.2. Path Diversity. We next evaluate the path diversity for SRFabric, which is defined as the number of disjoint shortest paths for each flow, rather than all of the possible paths. Higher path diversity offers a higher level of fault tolerance.

Figure 3(b) plots the path diversity achieved by all of the fabrics. The results show that 3D Torus achieves the highest path diversity compared with any other workloads, SRFabric performs the second, and finally followed by XFabric. This is because shorter path length sacrifices more possibilities of packet transmissions over disjoint paths, thus, leading to a lower path diversity. Obviously, the tradeoff between path length and path diversity is affected by the degree of reconfigurability. Fortunately, SRFabric ($d=2$) shows a higher diversity of 1.4 than 1.11 in XFabric for Map-Reduce, since SRFabric inherits the property of high path diversity from distributed fabrics. We further conclude that SRFabric behaves better on path diversity for Random workload than that for Map-Reduce and Graph Processing. The reason behind is that each SoC in Random workload shows urgent demands for direct connections. However, there are at most six direct links with one hop for each SoC, which limits the probability of reconfigurability and thus leads to a high path diversity. In the extreme case of all-to-all communications, reconfigurable topologies perform like that of static topologies.

4.2.3. Bisection Bandwidth. The bisection bandwidth of a network is the capacity across the “narrowest” bisector. A

higher bisection bandwidth implies a lower network oversubscription and better performance in term of the whole rack.

Figure 3(c) shows the average bisection bandwidth under various topologies. Note that all the results are normalized by the bisection bandwidth of a nonblocking network. 3D Torus delivers the lowest bisection bandwidth from 20% to 40%. Instead, SRFabric ($d=2$) delivers almost the same bisection bandwidth as that of XFabric for all of these workloads. For Map-Reduce, SRFabric provides 77% ($d=2$) and 68% ($d=1$) compared with the nonblocking bandwidth, while XFabric provides over 82% compared with the nonblocking bandwidth. Note that both SRFabric and XFabric can reconfigure the topology to adapt to various workload patterns. The small gap is incurred because SRFabric ($d=2$) only reconfigures two dynamic ports per SoC, while XFabric uses six circuit switches to feed these heavy communications.

Another important observation is that SRFabric, XFabric, and 3D Torus all behave very poor under random workload, resulting in only 24% of the nonblocking bandwidth. The reason is that these topologies are not able to provide adequate direct circuits to feed all these communications, leading to a higher path length and congestion in the intermediate SoCs.

To understand that, Figure 3(d) further shows the total network load on all of the links. Compared with 3D Torus, both SRFabric and XFabric deliver a lower network load. This is because both SRFabric and XFabric significantly decrease the path length and improve the network subscription by reconfiguration. For Map-Reduce, SRFabric ($d=2$) shows a similar network load with XFabric, i.e., 4.41, but the load decreases up to 40% than that of SRFabric ($d=1$). SRFabric has a higher average network link load up to almost 48 for random workload because a higher communication density per SoC weakens the reconfigurability. These results show that full link reconfigurability in XFabric does not translate into a significant improvement regarding the network load, as it costs 3 times more than that of SRFabric ($d=2$).

4.2.4. Impact of Rack Scales. We want to quantify the effect of reconfigurability and the effect of rack scale on the performance. This is accomplished by evaluating the average path length under 3 SRFabric configurations: ($d=1$), ($d=2$), and ($d=3$). We vary the rack size from 64 to 1024 and run the workload that is sampled randomly from the original trace.

Figure 4(a) plots the average path length under different SRFabric configurations and rack scales. It is observed that all the fabrics show a significant increase on the average path length as the number of SoCs increases. SRFabric ($d=1$) shows a similar distribution compared with 3D Torus because each SoC only has four static ports and one dynamic port. In the worst case with 1024 SoCs, SRFabric ($d=2$) behaves the worst regarding the average path length, i.e., 3.89 hops. SRFabric ($d=3$) comes second with 3.24 hops, followed by XFabric with 3.12 hops. It is expected that more dynamic ports per SoC offer a higher opportunity to cut down the source-destination path length. Although SRFabric ($d=3$) shows better performance than SRFabric ($d=2$), it uses 4

FIGURE 3: Results under various traffic patterns with $n = 512$ SoCs in a rack. (a) Average path length. (b) Average path diversity. (c) Average bisection bandwidth. (d) Rack throughput.

static ports and 3 dynamic ports per SoC, incurring additional port cost and wiring complexity. Therefore, SRFabric ($d=2$) is a better interconnection scheme for common workloads.

4.2.5. Impact of Communication Density. SRFabric performs poorly for random workload with higher density of traffic, as we have shown previously. To understand this more, we run the random workload iteratively and evaluate the average path length and bisection bandwidth by varying the number of SoC destinations per SoC from 2 to 512.

In Figure 4(b), it is observed that the average path length is stable, i.e., 6 hops, for 3D Torus because it is completely static with no reconfiguration. Given the same number of destinations per SoC, SRFabric ($d=1$) shows a worse performance on the average path length, while SRFabric ($d=2$) provides comparable performance with XFabric. For all-to-all communications, the average path length increases to 6 hops. This is because one dynamic port is enough to ensure a short average path length when only a small portion of inter-SoC traffic traverses through a multihops' path. Instead,

more dynamic links are required for a higher density of inter-SoC communications to cut down the path length.

Figure 4(c) shows the bisection bandwidth under various inter-SoC communication densities. Similarly, as the destinations per SoC increases, the network bisection bandwidth drops quickly to 0.16, except for 3D Torus. This is because a lot of inter-SoC flows use multihop paths and compete for the available bandwidth at bridge SoCs, decreasing the achievable bisection bandwidth. In addition, we observe almost the same trend in SRFabric ($d=2$) as that of XFabric. These results imply that the reconfigurability performs better in efficiently optimizing the inter-SoC traffic with low density.

4.2.6. Scalability of SRFabric. Assume that T_{node} is 460, and we now evaluate the performance of SRFabric over various link densities. Figure 5(a) shows the probability that the shortest path length is less than 4 hops. We conclude that our algorithm efficiently optimizes the dynamic ports and static ports for desired communication density and performance target. To ensure that the probability of at most 3-hops SPL is

FIGURE 4: Impact of various numbers of dynamic ports and communication peers per SoC. (a) Impact of SoC numbers. (b) Impact of destination. (c) Average bisection bandwidth.

above 90%, the communication density of links (324) only needs 3 dynamic ports, while that of links (3680) requires up to 6 ports.

We next evaluate the efficiency of Algorithm 1 by comparing it with modified XFabric that greedily exploits the dynamic ports to guarantee the performance. Figure 5(b) shows that SRFabric leads to a significantly lower port cost than that of XFabric for given communication density. As the density of links increases, the cost of SRFabric unexpectedly drops, while XFabric still keeps a great increase on the cost. When link density becomes high, the reconfigurability plays a little role on the performance improvement. The static ports are preferred to provide comparable performance like dynamic ports at a smaller cost. In the extreme case, SRFabric for all-to-all communications becomes a fully static topology.

To evaluate the relationship between dynamic ports and static ports, we vary the link density from 0.0001 to 1 and run SRFabric. Figure 5(c) illustrates the performance of SRFabric under various ports. When link density is less than 0.01, SRFabric only requires 9 ports and increases to 16 ports at

link density 0.1. When link density is larger than 0.6, SRFabric uses more cheap static ports rather than expensive dynamic ports. These results suggest that a higher level of reconfigurability incurs heavy cost than the improvement.

Finally, we evaluate the energy cost. Since XFabric [14] would require 0.4 kW to 0.6 kW versus a fully reconfigurable fabric using a folded Clos circuit switch that would require 2.2 kW, partial link reconfiguration consumes less than that of fully reconfiguration. Further shown in [29, 30], the per-port power draw ranges from 0.14 W for a typical optical circuit switch to 0.28 W for a 10 Gbps electrical circuit switch. Considering the number of ports in SRFabric, the cost is acceptable, which is less than 2.6 kW for a high density rack.

5. Related Work

Intra-datacenter (intra-DC) network contains two main parts: the networks between racks and the networks within a rack.

FIGURE 5: Scalability of SRFabric under various settings. (a) Performance of SRFabric. (b) Cost for both SRFabric and XFabric. (c) Nearly Optimal Number of Ports.

5.1. Networks upon ToRs. A number of data center topologies have been proposed in the literatures, such as DCell [31], fat tree [32, 33], and BCube [34]. All these static ToR-based topologies suffer from a high end-to-end delay in terms of the “eastern-western” traffic. Therefore, some noticeable efforts study the flexibility of intra-DC networks upon ToRs.

OSA [10], c-Through [11], and Helios [12] resort to the dynamic reconfigurability of optical circuit switches. Helios and c-Through achieve the flexibility via a limited number of single-hop optical links. OSA differs in which all the ToRs are connected to one large reconfigurable MEMS by multiple optical links. Considering the low cost of wireless transmissions, a novel method of dynamic reconfiguration using 60-GHz short-distance multi-Gigabit wireless links between ToRs is recently proposed by Kandula et al. [35, 36] to provide additional bandwidth for hot spots.

However, relatively high cost, power, and space requirements have limited the use of these ToR-based works for high-bandwidth and high-density in-rack networks.

Other works study the transmissions with high data rates via utilizing the advantages of flexible-grid elastic optical networking (EON). Lu et al. [37] discuss the technologies for realizing highly efficient data migration and backup for big data applications in elastic optical networks. The benefits incurred by EON for intra-DC networks have been discussed in [38, 39]. Li et al. [40] investigate how to select appropriate VNFs in intra-DC elastic optical networks.

Although these works improve the adaptivity of intra-DC networks through the management of switches and related spectrums, they fail to improve the in-rack interconnections since the transmissions within racks are also important.

5.2. Rack-Scale Computing. To provide high-speed and low-latency in-rack networks, considerable efforts have been invested into new high-speed network components, such as silicon photonics switches, Corning ClearCurve optical fiber, and MXC connector [41]. These new technologies make it

potential to address the bandwidth, distance, energy, and scalability challenges in-rack scale architecture [42–44].

There are some early examples regarding the rack scale computing, such as Boston Viridis [4], HP Moonshot [45], AMD SeaMicro 15000-OP [3], and Intel Rack-Scale Design [2]. Most of them employ a static multihop direct-interconnect distributed fabric, especially 3D Torus [46]. However, in face of diverse and unpredictable workloads in data center, these workload-specific designs no longer work well and have posed new challenges on routing and congestion control. Costa et al. rethink the network stack for rack scale computer [47] and propose a novel network control framework R2C2 [48] for flexible routing and efficient congestion control. Though R2C2 has made great efforts on improving the network utilization in network layer, traffic still suffers from serious delay due to a high average hops.

The closest one to ours is XFabric [14, 15] that enables all the links to be dynamically reconfigured and achieves fully link reconfigurability by attaching each port of a SoC to six crossbar ASICs. However, full link reconfigurability often does not translate into salient performance benefits as the high cost incurred. SRFabric addresses the tradeoff between cost, performance, and reconfigurability and presents the optimal design choice for the expected communication density and performance.

6. Conclusion

One promising trend of hyperscale data center is rack-scale computing, in which high density SoCs are integrated in the rack by high-bandwidth fabric. SRFabric is a semi-reconfigurable rack scale network that realizes partial link reconfigurability. We demonstrate SRFabric that can deliver higher path diversity and lower average path length at a reasonable cost, i.e., about 2 hops at only 29% cost of other algorithms. We also propose the algorithm that helps IT designers make the right decision on the reconfigurability according to the expected rack use and performance goals.

Data Availability

The data used to support the findings of the study are cited in [23].

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This work was supported by the Science and Technology Project of State Grid Corporation of China (Research and Application on Multi-Datcenters Cooperation & Intelligent Operation and Maintenance, no. 5700–202018194A-0-0-00).

References

- [1] “Tencent explores the datacenter using intel rack scale architecture,” 2019, <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/rack-scale-design-architecture-white-paper.pdf>.
- [2] “Intel rack scale design for DC,” 2017, <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/rack-scale-design-architecture-white-paper.pdf>.
- [3] Seamicro Sm15000,” <https://www.amd.com/en/press-releases/seamicro-sets-record-2014may13>.
- [4] “Boston viridis data sheet,” 2013, <https://www.boston.co.uk/solutions/viridis/default.aspx>.
- [5] M. Nanavati, J. Wires, and A. Warfield, “Isolation and sharing in disaggregated rack-scale storage,” in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation*, pp. 17–33, Boston, MA, USA, 2017.
- [6] Hp Proliant M800 server,” https://support.hpe.com/hpsc/public/docDisplay?docId=emr{text{_}na-c04500667.
- [7] “Dell poweredge c5220,” 2019, <https://www.dell.com/ky/business/p/poweredge-c5220/pd>.
- [8] K. Lotz, C. Su, Y. Zhang, and Y. Liu, “Structural analysis of three-dimensional mesh fabric by micro x-ray computed tomography,” *SAGE Journal of Engineered Fibers and Fabrics*, vol. 14, 2019.
- [9] M. S. Allen-Ware, J. Bloom, J. H. Chou et al., “Preparing computer nodes to boot in a multidimensional torus fabric network,” *SAGE Journal of Engineered Fibers and Fabrics*, vol. 10, 48.
- [10] K. Chen, A. Singla, A. Singh et al., “An optical switching architecture for data center networks with unprecedented flexibility,” *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 498–511, 2013.
- [11] G. Wang, D. G. Andersen, M. Kaminsky et al., “c-Through,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 327–338, 2010.
- [12] N. Farrington, G. Porter, S. Radhakrishnan et al., “Helios, ACM SIGCOMM computer communication review,” in *Proceedings of the ACM SIGCOMM 2010 Conference*, vol. 40, no. 4, pp. 339–350, New Delhi, India, 2010.
- [13] S. Tyler, W. Bernard, B. Tim, and R. Sajjad, “Towards reconfigurable rack-scale networking,” *WRSC*, 2015.
- [14] S. Legtchenko, N. Chen, D. Cletheroe et al., “Xfabric: a reconfigurable in-rack network for rack-scale computers,” in *Proceedings of the NSDI’16*, pp. 15–29, Santa Clara, CA, USA, 2016.
- [15] O. S. Sella, A. W. Moore, and N. Zilberman, “High speed adaptive rack-scale fabrics,” in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, pp. 33–35, Budapest, Hungary, 2018.
- [16] Y. Zhang, X. Xiao, K. Zhang et al., “Foundry-enabled scalable all-to-all optical interconnects using silicon nitride arrayed waveguide router interposers and silicon photonic transceivers,” *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 25, no. 5, pp. 1–9, 2019.
- [17] C.-H. Wang and T. Javidi, “Adaptive policies for scheduling with reconfiguration delay: an end-to-end solution for all-optical data centers,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1555–1568, 2017.
- [18] “Torus interconnect,” 2020, https://en.wikipedia.org/wiki/Torus{text{_}interconnect.
- [19] K. M. Imre, C. Baransel, and H. Artuner, “Efficient and scalable routing algorithms for collective communication operations on 2d all-port torus networks,” *International Journal of Parallel Program.* vol. 39, no. 6, pp. 746–782, 2011.
- [20] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, “Distributed SDN controller system: a survey on design choice,” *Computer Networks*, vol. 121, pp. 100–111, 2017.
- [21] T. Wang, F. Liu, and H. Xu, “An efficient online algorithm for dynamic sdn controller assignment in data center networks,”

- IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2788–2801, 2017.
- [22] K. Manjula and S. M. Sundaram, “Optimized approach (SPCA) for load balancing in distributed HDFS cluster,” *SN Comput., Sci.*, vol. 1, no. 2, p. 102, 2020.
 - [23] A. Sabne, P. Sakdhnagool, and R. Eigenmann, “A mapreduce programming system for accelerator clusters,” in *Proceedings of the ACM 24th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 235–246, 2015.
 - [24] B. Bollobás and B. Béla, *Random Graphs*, p. 73, Cambridge University Press, Cambridge, UK, 2001.
 - [25] V. D. Blondel, J.-L. Guillaume, J. M. Hendrickx, and R. M. Jungers, “Distance distribution in random graphs and application to network exploration,” *Physical Review E*, vol. 76, no. 6, Article ID 066101, 2007.
 - [26] E. Katzav, M. Nitzan, D. ben-Avraham et al., “Analytical results for the distribution of shortest path lengths in random networks,” *EPL (Europhysics Letters)*, vol. 111, no. 2, p. 26006, 2015.
 - [27] C.-M. Cheung and K.-C. Leung, “DFFR: a flow-based approach for distributed load balancing in Data Center Networks,” *Computer Communications*, vol. 116, pp. 1–8, 2018.
 - [28] M. Alam, K. A. Shakil, and S. Sethi, “Analysis and clustering of workload in google cluster trace based on resource usage,” in *Proceedings of the 15th International Symposium on Distributed Computing and Applications for Business Engineering*, pp. 740–747, Paris, France, August 2016.
 - [29] Calient Optical Circuit Switch, 2020, <http://www.calient.net/download/s320-optical-circuit-switch-datasheet>.
 - [30] Macom M21605 Crosspoint Switch Specification, <http://www.macom.com/products/product-detail/M21605/>.
 - [31] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, “Dcell,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 75–86, 2008.
 - [32] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.
 - [33] R. Niranjan Mysore, A. Pamboris, N. Farrington et al., “PortLand,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 39–50, 2009.
 - [34] C. Guo, G. Lu, D. Li et al., “BCube,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 63–74, 2009.
 - [35] S. Kandula, J. Padhye, P. Bahl et al., *Flyways to De-congest Data Center Networks*, HOTNETS, New York, NY, USA, 2009.
 - [36] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, “Augmenting data center networks with multi-gigabit wireless links,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 38–49, 2011.
 - [37] P. Lu, L. Zhang, X. Liu, J. Yao, and Z. Zhu, “Highly efficient data migration and backup for big data applications in elastic optical inter-data-center networks,” *IEEE Network*, vol. 29, no. 5, pp. 36–42, 2015.
 - [38] S. B. Yoo, Y. Yin, and K. Wen, “Intra and inter datacenter networking: the role of optical packet switching and flexible bandwidth optical networking,” in *Proceedings of the IEEE 16th International Conference on Optical Network Design and Modelling*, pp. 1–6, Rome, Italy, 2012.
 - [39] L. Peng, M. Chen, K. Park, and C.-H. Youn, “Virtual-pod-assisted routing and resource assignment in elastic all-optical intra-datacenter networks,” *IEEE Access*, vol. 5, pp. 406–420, 2016.
 - [40] Y. Li, Y. Zhao, B. Li et al., “Joint balancing of it and spectrum resources for selecting virtualized network function in inter-datacenter elastic optical networks,” *Optical Society of America Optics Express*, vol. 27, no. 11, pp. 15 116–215 128, 2019.
 - [41] M. Iwaya, “High-density multi-lane connectivity for optical interconnect,” *IEEE*, pp. 160–161, 2015.
 - [42] H. Volos, K. Keeton, Y. Zhang et al., “Memory-oriented distributed computing at rack scale,” in *Proceedings of the ACM Symposium on Cloud Computing*, p. 529, New York, NY, USA, 2018.
 - [43] S. Novakovic, A. Daglis, D. Ustiugov, E. Bugnion, B. Falsafi, and B. Grot, “Mitigating load imbalance in distributed data serving with rack-scale memory pooling,” *ACM Transactions on Computer Systems*, vol. 36, no. 2, pp. 1–37, 2019.
 - [44] V. Eramo, M. Ammar, and F. G. Lavacca, “Migration energy aware reconfigurations of virtual network function instances in nfv architectures,” *IEEE Access*, vol. 5, pp. 4927–4938, 2017.
 - [45] “Hp moonshot system,” 2020, <https://www.hpe.com/us/en/servers/moonshot.html>.
 - [46] P. Costa, A. Donnelly, G. O’Shea et al., “Camcubeos: a key-based network stack for 3d torus cluster topologies,” in *Proceedings of the HPDC’13*, pp. 73–84, New York, NY, USA, 2013.
 - [47] P. Costa, H. Ballani, and D. Narayanan, “Rethinking the network stack for rack-scale computers,” in *HotCloud*, 2014.
 - [48] P. Costa, H. Ballani, K. Razavi, and I. Kash, “R2C2: a network stack for rack-scale computers,” *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 551–564.