GRAP: Group-level Resource Allocation Policy for Reconfigurable Dragonfly Network in HPC

Guangnan Feng
fenggn3@mail2.sysu.edu.cn
Sun Yat-sen University
Guangzhou, China

Dezun Dong
dong@nudt.edu.cn
College of Computer
National University of Defense Technology
Changsha, China

Shizhen Zhao
shizhenzhao@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Yutong Lu*
luyutong@mail.sysu.edu.cn
Sun Yat-sen University
Guangzhou, China

ABSTRACT

Dragonfly is a highly scalable, low-diameter, and cost-efficient network topology, which has been adopted in new exascale High Performance Computing (HPC) systems. However, Dragonfly topology suffers from the limited direct links between groups. The reconfigurable network can solve this problem by reconfiguring topology to adjust the number of direct links between groups. While the performance improvement of a single job on reconfigurable HPC network has been evaluated in previous works, the performance of HPC workloads has not been studied because of the lack of an appropriate resource allocation policy.

In this work, we propose Group-level Resource Allocation Policy (GRAP) to allocate both compute nodes and Reconfigurable Links for jobs in Reconfigurable Dragonfly Network (RDN). We start with formulating three design principles: reconfigurable network should be reconfiguration interference-free, guarantee connectivity and performance for each job, and satisfy varied resource requests. According to the principles, GRAP uses different strategies for small and large jobs, and contains three allocation modes for large jobs: Balance Mode, Custom Mode, and Adaptive Mode. Finally, we evaluate GRAP with the CODES network simulation framework and the Slurm Simulator using real workload traces. The results demonstrate that RDN coupled with GRAP achieves lower latency, higher bandwidth, and lower job wait time.

CCS CONCEPTS

• **Networks** → **Network resources allocation**.

KEYWORDS

Reconfiguable Network, Dragonfly, HPC, Resource Allocation

*Corresponding author

ICS '23, June 21–23, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0056-9/23/06…$15.00
https://doi.org/10.1145/3577193.3593732

ACM Reference Format:
Guangnan Feng, Dezun Dong, Shizhen Zhao, and Yutong Lu. 2023. GRAP: Group-level Resource Allocation Policy for Reconfigurable Dragonfly Network in HPC. In *2023 International Conference on Supercomputing (ICS '23), June 21–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3577193.3593732

1 INTRODUCTION

The Dragonfly topology[30] is one of the most popular topologies for highly scalable, low-diameter, and cost-efficient interconnection networks. It has been adopted in many High Performance Computing (HPC) systems, including Piz Daint[11] and Trinity[31] with the Aries interconnect[18], Frontier[37] and Perlmutter[46] with the Slingshot interconnect[16].

The Dragonfly topology is a two-tiered direct topology consisting of groups of switches connected to compute nodes. A key characteristic of Dragonfly topology is that it has at least one direct global link between any two groups. Such global links provide low diameter and high scalability but cause congestion when too much data is transferred between two groups[22].

The reconfigurable network can solve this problem by reconfiguring topology to adjust the number of direct links between groups. The Reconfigurable Dragonfly Network(RDN) takes advantage of the Optical Circuit Switch (OCS), which can establish and remove an optical data path between any input and output port according to the traffic pattern. It can alleviate the congestion in Dragonfly topology[64].

The reconfigurable network has been adopted in Google's data centers [52] but has never been deployed in a production HPC system[1]. In previous works[4, 13, 35, 38, 42, 58, 64, 69], the performance improvement of a single job on the reconfigurable HPC network has been evaluated, but the resource allocation policy for HPC workloads is ignored or designed unrealistically. And the resource allocation policy for traditional networks cannot be used in reconfigurable networks because it does not consider the allocation of Reconfigurable Links, which will cause connectivity and performance problems. The lack of appropriate resource allocation policy for reconfigurable networks leads to unknown performance of HPC workloads on reconfigurable networks.

[1]This work was finished in February 2023. Google published its new supercomputer system for AI with reconfigurable network in Apr. 2023[29].

In this work, we aim to design a resource allocation policy to allocate both compute nodes and Reconfigurable Links for jobs in RDN. According to our experience with real supercomputing systems, we first formulate three design principles for RDN:

(1) Reconfiguration should not interfere with running jobs;
(2) Connectivity and performance should be guaranteed for each job;
(3) Varied resource requests should be satisfied.

Based on these design principles, we design Group-level Resource Allocation Policy (GRAP). GRAP uses different strategies and restrictions for small and large jobs, and contains three allocation modes for large jobs: Balance Mode, Custom Mode, and Adaptive Mode. Each mode has its target traffic patterns and further resource allocation rules.

We design a two-step experiment to evaluate the performance of HPC workloads on the Reconfigurable Dragonfly Network coupled with the Group-level Resource Allocation Policy (RDN-GRAP). We first use CODES network simulation framework[45] to evaluate the three resource allocation modes with corresponding traffic patterns under different job sizes. Then, we use the results as input to evaluate the performance of real HPC workloads from Mira supercomputing system[14, 19] with Slurm Simulator[54] and compared it with Traditional Dragonfly Network (TDN).

The key contributions of this work can be summarized as follows:

- We formulate three design principles for resource allocation policy in RDN according to our experience with real supercomputing systems.
- We design a resource allocation policy GRAP which allocates both compute nodes and Reconfigurable Links for jobs in RDN according to the design principles. It is compatible with many existing designs for Reconfigurable HPC Networks, including the routing and reconfiguration algorithms.
- We perform detailed performance evaluations for RDN-GRAP, and the results demonstrate that RDN-GRAP achieves inter-job interference-free, lower latency, higher bandwidth, and lower job wait time compared to TDN.

The rest of this paper is organized as follows: Section 2 introduces Dragonfly topology and RDN. In Section 3, we formulate design principles for RDN. In Section 4, we introduce GRAP. In Section 5, we evaluate RDN-GRAP and compare it with TDN. Finally, Section 6 introduces related works and Section 7 presents our conclusions.

2 BACKGROUND AND MOTIVATION

2.1 Dragonfly Topology

A standard Dragonfly topology has a 2-layer structure[30]. A group of switches are interconnected using an intra-group topology into a group that can be regarded as a single virtual switch with a very high radix. The groups are then fully connected with an inter-group topology. As to intra-group topology, different topologies can be used[16, 18]. In this work, we focus on the traffic among groups and use full-mesh topology for intra-group, where all switches are directly connected to each other within a group. GRAP, however, can be applied to other intra-group topology variations.

The number of groups in a Dragonfly can vary. The largest possible Dragonfly has only one global link connecting each pair of groups. Three parameters uniquely define the largest Dragonfly: the number of links per switch connecting to compute nodes p, the number of switches in each group a, and the number of global links per switch connecting to switches in other groups h. As discussed in [30], a load-balanced Dragonfly system should satisfy $a = 2p = 2h$.

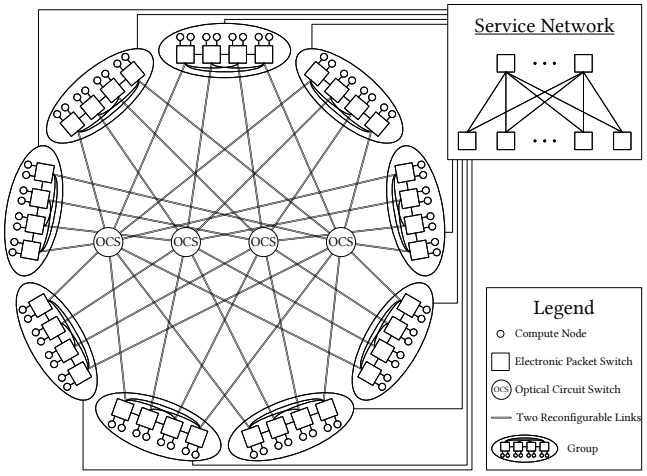


Figure 1: Reconfigurable Dragonfly Network.

Since each group connects to the other groups evenly, Dragonfly topology works well in uniform traffic. However, when a large amount of data is transferred between two groups, the limited number of direct links between two groups will lead to congestion[22]. To address this issue, two solutions are proposed. The first one is adaptive routing[27, 55]. Packets are detoured to other groups when congestion happens in the minimal paths. The second one is random/round-robin job placement that spreads the traffic over the whole network evenly[8, 9, 25].

However, both solutions have their limitations. Adaptive routing will cause higher average hops and lower network throughput when non-minimal paths are selected frequently. In addition, adaptive routing needs a more complicated network routing chip design. The discontinuous job placement loses the performance of neighboring communication[33, 60]. Zahn et al. show that neighbor communication is frequently used in HPC applications[68]. The discontinuous job placement makes intra-group communication becomes inter-group communication. Even worse, it will cause inter-job interference[66].

2.2 Reconfigure Dragonfly Network

In this subsection, we introduce the Reconfigurable Dragonfly Network, a combination of the advantages of several previous work[4, 57–59, 64]. It is the network where GRAP is based.

RDN can be seen as a traditional Dragonfly topology with all its global links connected to OCSes, as shown in Figure 1. It is similar to [4, 64], but a little different. Each group's ith global links are connected to the same OCS. We call them Reconfigurable Links in this work. We expect the number of ports of an OCS to be larger than the number of groups in the Dragonfly topology. OCS with 320 ports is currently available[1]; meanwhile, 512 and 1024 ports MEMS-based OCSes are on their way. If the number of ports of

OCS is adequate, each group can connect more Reconfigurable Links to the same OCS. In Figure 1, we let each group connects two Reconfigurable Links to the same OCS.

Reconfigurable networks take advantage of the OCS, which can establish and remove an optical data path between any input and output port. According to traffic patterns, OCSes are configured to disconnect the optical data paths between groups with little communication and establish optical data paths between groups that communicate heavily.

Storage access, connectivity, and fault tolerance are easily overlooked problems in reconfigurable network design. To overcome these problems, we prefer to add a separate service network to assist reconfigurable network, which is similar to [4, 20, 61]. The separate network is made of Electronic Packet Switches (EPSes) and is used for service. The service network can adopt any topology. A highly oversubscribed Fat-tree topology is suggested to save budget and power. The service network in Figure 1 is an example. Each EPS connecting to compute nodes is connected to a service network's leaf switch by a single link. Compute nodes can access storage by the service network at any time, regardless of the current status of the reconfigurable topology. Similarly, the system administrator, Slurm service, and users can access compute nodes anytime. When OCS failures occur, the traffic can temporarily fall back to the service network and let jobs be suspended without timeout error. Therefore, our resource allocation policy does not need to consider the storage access, compute nodes access, and fault-tolerant problem.

Topology-Aware, Globally-direct Oblivious (TAGO) routing[57] is adopted for inter-group routing in RDN. To our knowledge, this is the only study focused on routing for reconfigurable networks. The inter-group traffic is load balanced among all the direct links from the source group to the destination group. Indirect routing that detours from another group is not required because the topology has matched the traffic pattern. For intra-group routing, we adopt adaptive routing to prevent contention in intra-group communication.

Previous works[4, 13, 35, 58, 64] have studied the traffic pattern of HPC applications and propose topology reconfiguration schemes to match applications' communication patterns. These schemes include adaptive schemes[4], schemes obtained by solving min-max decomposition problem[64] and min-cost flow problem[58], and schemes based on Machine Learning (ML)[13, 35]. In this work, GARP is compatible with all these reconfiguration schemes, which is one of its advantages.

The resource allocation policy is ignored or assumed unrealistically in previous work. Some of them only consider a single job running in the HPC system[4, 35, 43, 59, 64]. However, in a real HPC system, many users use the system simultaneously, and many jobs are executed at the same time. [57] considers mixed traffic patterns from different applications and compares contiguous job mapping and randomized job mapping, but the resource allocation policy for jobs that will arrive and finish differently is not considered. [58] underestimates the interference of reconfiguration on running jobs and assumes that topology can be reconfigured every time a new job starts. In addition, the resource allocation policy for traditional networks cannot be used in reconfigurable networks because it does not consider the allocation of Reconfigurable Links. If the Reconfigurable Links are not appropriately allocated, it will cause connectivity and performance problems. A job may not be able to start because its nodes cannot communicate with each other. And the performance of some jobs may decrease because of insufficient Reconfigurable Links. These problems will be analyzed in detail in Principle 2.

In this paper, we do not analyze RDN's cost in detail, including capex and opex, since it will be too speculative. We strongly expect that OCSes in Reconfigurable Dragonfly Network will not add much extra cost. In capex, a single OCS contains hundreds of ports and needs no extra Optical Transceiver Modules. In opex, the power consumption of the current 320 ports OCS is only 45 watts[1].

3 DESIGN PRINCIPLES

In this section, we formulate the three design principles for RDN in HPC and explain the reasons.

Principle 1. ***Reconfiguration should not interfere with running jobs.***

HPC systems usually adopt the lossless network and Remote Direct Memory Access (RDMA) to achieve ultra-low latency and high performance. If a link with traffic is disconnected, it will cause packet loss and Go-Back-N error recovery, leading to increasing tail latency. If one process is blocked by packet retransmission, the slowdown may propagate to all processes in this job. If a new routing path is not established in time, the packets may keep waiting in the buffer of intermediate switches until a timeout and cause network congestion. Even worse, not only the jobs that use the disconnected link may be stuck, other jobs affected by the congestion may undergo a performance decrease.

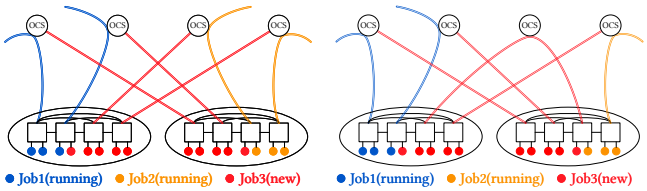
Topology reconfigurations, especially for disconnecting links, should consider the communication requirements of all jobs to avoid victim jobs. Topology reconfiguration should not break the connectivity of any job. The links in the only path of two nodes in the same job should never be disconnected until the job finishes or a new path is available. Further, disconnections should be decided carefully to avoid rerouting traffic to a far or congested path.

As the jobs in HPC system start and finish, the traffic pattern in the network changes rapidly. It means the mechanisms that prevent network reconfiguration from interfering with running jobs should be fast enough to deal with frequent network reconfiguration.

This problem can be dealt with a complicated topology reconfiguration algorithm and rerouting scheme. However, we solved it with the resource allocation policy GRAP, which dramatically reduces the complexity of the reconfiguration algorithm and requires no rerouting scheme.

Principle 2. ***Connectivity and performance should be guaranteed for each job.***

The connectivity and performance problems are critical issues in the resource allocation policy of reconfigurable HPC networks. Guaranteeing the connectivity of a job means that all the nodes allocated for the job can reach each other after topology reconfiguration. Figure 2a is an example that the connectivity of Job3 is not guaranteed. Job1 and Job2 are running jobs with allocated Reconfigurable Links. Their allocated Reconfigurable Links are connected to other groups. Job 3 is a new job. Suppose the red nodes



(a) Connectivity Problem (b) Performance Problem

Figure 2: Possible connectivity and performance problems.

and Reconfigurable Links are allocated for Job3. In that case, the Reconfigurable Links cannot be configured to form a connected graph because the Reconfigurable Links are connected to different OCSes. Thus, the nodes in Job3 cannot communicate with nodes in another group. Reconfiguring the Reconfigurable Links of Job1 and Job2 should be cautious because it may violate Principle 1. Thus, some mechanisms must be designed to avoid the connectivity problem.

Guaranteeing the performance of a job means that appropriate and enough Reconfigurable Links are allocated for the job. Figure 2b is another example that Job3 may have a performance problem. If the red nodes and Reconfigurable Links are allocated for Job3, the nodes in the two groups are connected but connected by only two global links (the red curve represents two Reconfigurable Links). The other Reconfigurable Links are wasted because they are not connected to the same OCSes. Thus, the bandwidth between the two groups is limited. If there is a lot of traffic between the two groups in Job3, the performance of Job3 will decrease. Thus, some mechanisms must be designed to guarantee performance for each job.

Principle 3. ***Varied resource requests should be satisfied.***

Although many previous works take the traced traffic patterns as the input of topology reconfiguration algorithms, they did not consider users' configuration can change that traffic patterns. We advocate that users who run large jobs should adjust the configuration of their jobs to match the reconfigurable network topology to achieve the best performance. Just like the users of structured grid computing in the Torus network who usually decompose their grid in each dimension to adapt the size of the Torus topology, which not only load balance calculation but also take advantage of the neighboring communication performance of Torus topology. The resource allocation system of the Torus network provides users with an interface for specifying the used size in each dimension.

In an RDN, users pursuing the best performance should understand the basic concept of the network: The network topology has many groups, and each group contains x compute nodes. The intra-group topology provides better communication performance than the inter-group topology. Thus, they should try to adjust the communication to intra-group communication under the limitation of x nodes in each group. As for the inter-group topology, it is reconfigurable. The direct links between two groups can be increased if needed. However, the number of inter-group links is limited. The inter-group links should be placed at traffic hot spots. Thus, users should simplify the inter-group communication pattern to let each group communicate with as few groups as possible. Such changes

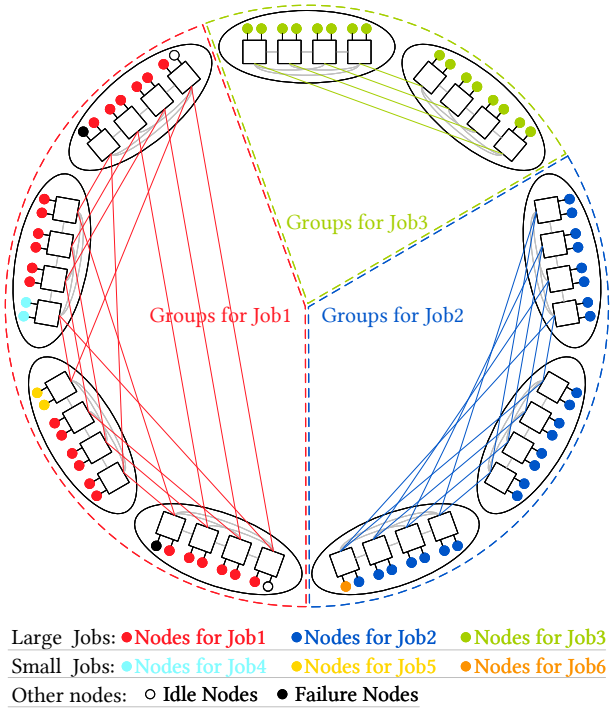


Figure 3: Job Classification and Restriction in GRAP: A small job is always allocated nodes within a single group without Reconfigurable Links. Each large job will be allocated nodes from several groups that do not contain other large jobs, and all the Reconfigurable Links attached to these groups.

in communication patterns should be achieved by modifying only the input parameters rather than the source code.

System designers must provide users with a resource allocation mode interface to let jobs use the expected resource, and each process is placed on the correct node. A good approach is increasing parameters in the resource request command line. In addition, a default setting is necessary for users who know nothing about RDN and its resource allocation mode interface. The system also needs to guarantee the performance of their job.

4 GRAP DESIGN AND IMPLEMENTATION

Resource allocation policy is a part of the Resource and Job Management System (RJMS). Resource allocation policy takes a job's resource request and currently available resources as input. If the currently available resources are adequate, it will return a subset of the resources that satisfies the request. RJMS also contains an ordering policy that reorders the jobs in queue and a reservation policy that determines whether to reserve specific resources for future jobs. Our work only designs the resource allocation policy part of RJMS.

In GRAP, all jobs are first classified into small and large jobs. Then, each large job can use one of three resource allocation modes to fit into its traffic pattern. We will introduce these two steps in the first two subsections. In the last subsection, we further analyze and discuss the characteristics of GRAP.

4.1 Job Classification and Restriction

In GRAP, all the incoming jobs are classified into small and large jobs based on the number of nodes requested, as shown in lines 1-5 of Algorithm 1. The small and large jobs have different resource allocation restrictions.

- **Small Job**: The number of nodes requested by a small job is fewer than or equal to the number of nodes in a group. A small job is always allocated nodes within a single group without Reconfigurable Links.
- **Large Job**: The number of nodes requested by a large job is larger than the number of nodes in a group. It spans multiple groups. Each group contains at most one large job. Each large job is allocated not only the nodes it requires but also all the Reconfigurable Links of the groups that contain nodes for this job.

The key design of GRAP is that all the Reconfigurable Links of a group belong to the only large job in this group. This key design follows the Principle 1. Jobs will not affect each other during the reconfiguration of topology. The small jobs are restricted to nodes within a single group, so they do not require Reconfigurable Links. The start and finish of small jobs won't trigger any network reconfiguration. Each large job will be allocated nodes from several groups that do not contain other large jobs, and all the Reconfigurable Links attached to these groups. According to the job's traffic pattern, the groups owned by this job are connected to each other by a traffic-matched topology before the start of execution. Thus, both topology reconfiguration and communication traffic of large jobs will not affect each other because groups and global links are not shared among large jobs.

This key design also satisfies the Principle 2. The connectivity is guaranteed for each job, and GRAP provides the best network performance for each job. Small jobs can take advantage of intra-group communication performance and do not have a connectivity problem. As for any large job, the groups that contain nodes for this job can be seen as a smaller RDN because all the Reconfigurable Links attached to these groups belong to this job. Since no more useful Reconfigurable Links can be allocated to this job, it is the best network performance that can be provided for this job. Obviously, there is no connectivity problem for large jobs either.

Figure 3 is an example of GRAP. Job1, Job2, and Job3 are large jobs. Job1 is allocated with four groups, and Job2 is allocated with three groups. These groups are connected to form two smaller Dragonfly subnets. This is the default topology for large jobs. The number of direct global links between two groups in these smaller Dragonfly subnets is increased compared to the Dragonfly formed by all the groups. The performance of inter-group communication will improve. Job3 are allocated with two groups. All the Reconfigurable Links are directly connected to another group in the job. The number of global links between two groups is dramatically increased compared to only one direct global link between two groups in Dragonfly formed by all the groups. Job4, Job5, and Job6 are small jobs. They consume the fragmented resources produced by large jobs.

The allocation policy for small jobs is shown in lines 6-18 of Algorithm 1. GRAP will first filter the groups with enough idle nodes for the current small job (line 7). If no such group exist, the current small job has to wait in a queue(line 8-9). If such a group exists, GRAP will consider groups that already contain a large job first (lines 12-13) and then those with no large job second (lines 14-15). Then, GRAP will choose the group with the least idle nodes for the current small job (line 13 or 15). Note that the allocation policy of selecting nodes within a group (line 16) is not a part of this work because Dragonfly's intra-group topology can vary according to its definition. This is the same in large job allocation policy.

Algorithm 1: **G**roup-level **R**esource **A**llocation **P**olicy

```
Input: Job: details of current job,
Df: details of topology including available resource.
Output: Job_res: nodes and Reconfigurable Links for Job
Data: _*:built-in indicator, N*:number of *, idle:idle nodes,
can_grps: candidate groups after selection
1  Function GRAP(Job, Df)
2      if if Job.Nnodes ≤ Df.Nnodes_in_group then
3          return Small_Job_Allocation(Job, Df);
4      else
5          return Large_Job_Allocation(Job, Df);
6  Function Small_Job_Allocation(Job, Df)
7      can_grps = select(Df.groups, _idle ≥, Job.Nnodes);
8      if can_grps.size == 0 then
9          return _JOB_WAIT;
10     can_grps_with = select(Df.groups, _with_large_job);
11     can_grps_no = select(Df.groups, _no_large_job);
12     if can_grps_with.size ≠ 0 then
13         target_grp = min(can_grps_with, _by_Nidle);
14     else
15         target_grp = min(can_grps_no, _by_Nidle);
16     Job_res.nodes = select(target_grp.idle, Job.Nnodes);
17     Job_res.links = _none;
18     return Job_res;
```

This job classification and restriction may cause longer job wait time in some cases. Nevertheless, the performance improvement brought by RDN-GRAP can mitigate this issue, as we will show in Section 5.3.

In real production supercomputing systems, large jobs consume much more core-hours than small jobs. Patel et al. investigate the job characteristics of the Mira supercomputing system, which contains 49152 compute nodes. Their analysis shows that nearly 80% of core hours were consumed by jobs that require more than 4k nodes in 2018[49]. According to our experience, small jobs are usually created by developers and students for testing, debugging, and studying. For small jobs, many systems have a time limitation, for example, one hour. Thus, small jobs are suitable for consuming the fragmented resources created by large to improve system utilization.

Another potential benefit of this design is preventing users from forming a large resource set by submitting a series of smaller resource requests. It will cause a deadlock in resource allocation when multiple users hold resources and wait for more resources. If they do, they will find out that the nodes in different groups may not be able to communicate with each other.

Algorithm 2: Large_Job_Allocation

Input: *Job*: details of current job
Df: details of topology including available resource
Output: *Job_res*: nodes and Reconfigurable Links for *Job*
Data: _∗: built-in indicator, *N*∗: number of ∗, *idle*:idle node,
can_grps: candidate groups after selection,
Ngrps: number of groups used for *Job*,
node_reqs: number of nodes requested in each group for *Job*

```
1  Function Large_Job_Allocation(Job, Df)
2      can_grps = select(Df.groups, _no_large_job);
3      if Job.mode == _MINIMAL_BALANCE then
4          Ngrps = Job.Nnodes/Df.Nnodes_in_group;
5          node_reqs[1...Ngrps] = Df.Nnodes_in_group;
6      else if Job.mode == _CUSTOM_BALANCE then
7          Ngrps = Job.user_defined_Ngroup;
8          node_reqs[1...Ngrps] = Job.num_nodes/Ngrps;
9      else if Job.mode == _CUSTOM then
10         Ngrps = Job.user_defined_Ngroup;
11         for i = 1 to Ngrps do
12             node_reqs[i] = Job.user_node_reqs[i];
13     else if Job.mode == _ADAPTIVE then
14         return adaptive_mode_allocation(Job, Df);
15     sort(can_grps, _by_Nidle, _ascend);
16     sort(node_reqs, _by_Nidle, _ascend);
17     j = 1;
18     for i = 1 to Ngrps do
19         while j ≤ can_grps.size and can_grps[j].Nidle <
            nnode_reqs[i] do
20             j += 1;
21         if j > can_grps.size then
22             return _JOB_WAIT;
23         else
24             Job_res.nodes += select(can_grps[j].idle,
                nnode_reqs[i]);
25             Job_res.links += can_grps[j].all_ocs_links;
26     return Job_res;
```

4.2 Allocation Modes

The allocation policy for large jobs is shown in Algorithm 2. GRAP filters the groups that contain no large job as candidate groups (line 2) and then allocates nodes for the current large job according to the job's allocation mode specified by the user when submitting the job. This design follows the Principle 3 to provide users with a resource allocation interface.

4.2.1 Balance Mode

Balance Mode is used for jobs requiring equal numbers of nodes in each group. The traffic pattern of these jobs usually stresses intra-group communication. The intra-group topology of Dragonfly can provide lower latency and better performance for these jobs. Taking large-scale deep learning training as an example, pipeline parallelism and operator parallelism can be used within a single group, while Data parallelism can be used among groups. Each group holds a copy of the model[6].

Algorithm 3: Adaptive_Mode_Allocation

Input: *Job*: details of current job
Df: details of topology including available resource
Output: *Job_res*: nodes and Reconfigurable Links for *Job*
Data: _∗: built-in indicator, *N*∗: number of ∗, *idle*:idle node,
can_grps: candidate groups after selection

```
1  Function Adaptive_Mode_Allocation(Job, Df)
2      can_grps = select(Df.groups, _no_large_job);
3      sort(can_grps, _by_Nidle, _descend);
4      Ngrps = ⌈Job.Nnodes/Df.Nnodes_in_group⌉;
5      st = 1;
6      en = Ngrps;
7      if sum(can_grps[st...en].Nidle) < Job.Nnodes then
8          return _JOB_WAIT;
9      while en + 1 ≤ Df.Ngroups and
        sum(can_grps[st + 1...en + 1].Nidle) ≥ Job.Nnodes do
10         st += 1;
11         en += 1;
12     Nnodes_remain = Job.Nnodes;
13     for i = en to st do
14         Nnodes_req = Nnodes_remain/(i − st + 1);
15         if Nnodes_req > can_grps[i].Nidle then
16             Nnodes_req = can_grps[i].Nidle;
17         Job_res.nodes +=
            select(can_grps[i].idle, Nnodes_req);
18         Job_res.links += can_grps[i].all_ocs_links;
19         Nnodes_remain -= nodes_req;
20     return Job_res;
```

Balance Mode is further divided into two sub-mode: Minimal Balance Mode and Custom Balance Mode.

In Minimal Balance Mode (lines 3-5), the number of groups used by this large job, *Ngrps*, is set to *Job.Nnodes*/*Df.Nnodes_in_group* (the number of nodes requested by the current large job divides the number of nodes in a group). In this sub-mode, we require the job can be configured to use all the nodes in each allocated group. For example, in solvers for incompressible turbulent flows, such as Incompact3D[32], CaNS[15], and PowerLLEL[65], the 3D cartesian grid can be decomposed arbitrarily in two dimensions (denoted as row and column). The number of nodes used in each row or column can be set to *Df.Nnodes_in_group*.

As for Custom Balance Mode (lines 6-8), *Ngrps* is specified by the user. The number of nodes used in each group is set to *Job.Nnodes*/*Ngrps*. This sub-mode is designed for jobs where the number of nodes in each allocated group cannot be arbitrary. For example, the number of processes for Quantum Exact Simulation Toolkit (QuEST)[28] must be the power of two.

4.2.2 Custom Mode

Custom Mode (lines 9-12) is used for jobs where the number of nodes in each group needs to be different. Users need to specify not only *Ngrps* but also the number of nodes used in each group. This mode is designed to cover all special cases. For example, Community Earth System Model (CESM)[2] is an application that contains

multiple modules. Each module may be executed by different numbers of nodes in parallel. This may lead to a different number of node requirements in each group.

In Balanced Mode and Custom Mode, the number of nodes required in each group is specific. After sorting the node requirements and the candidate groups in ascending order (lines 15-16), GRAP tries to find a group that contains enough idle nodes for each requirement one by one to reduce fragmentation resources, as shown in lines 17-25.

4.2.3 Adaptive Mode

Adaptive Mode is the default mode for large jobs if a user does not explicitly specify Balance Mode or Custom Mode. The Adaptive Mode lowers the difficulty of using the RDN. The performance of many applications is not sensitive to the number of nodes used in each group. For example, the uniform traffic is not sensitive to the number of nodes used in each group, which is proved in Section 5.2. The algorithm used for Adaptive Mode is shown in Algorithm 3. GRAP still uses as few groups as possible and set $Ngrps$ to $\lceil Job.Nnodes/Df.Nnodes\_in\_group \rceil$ because groups that do not contain large jobs are regarded as rare resources, which is similar to Minimal Balance Mode. The difference is the number of nodes used in each group is not required to be equal but should be as close as possible. Then, GRAP sorts the candidate groups in descending order and chooses $Ngrps$ continuous groups that can satisfy the current job from sorted candidate groups. And the $Ngrps$ continuous groups should be chosen as far back as possible (lines 9-11) to preserve more groups with more idle nodes.

4.3 Further analysis and discussions

4.3.1 Keep It Simple and Smart

In real HPC systems, the resources are managed by both RJMS and the human administrator. It will be better for the resource allocation policy to be more concise and understandable to avoid trouble with the system administrator.

There are multiple resource pools for different levels of users in an HPC system. The administrator needs to assign nodes to resource pools, set users' priorities for different resource pools, and dynamically adjust them according to actual usage. Administrators need to understand the resource allocation policy and cooperate with it. For example, in the Fat-tree topology, the administrator preferred to assign nodes in the same sub-tree to the same resource pools to provide lower latency and better performance. In RDN-GRAP, resources can be managed and assigned to resource pools in groups.

Sometimes, the resource allocation policy is executed by not only RJMS but also the administrator. For some emergency jobs, such as the virus analysis when COVID-19 outbroke, some resources are reserved for them to use exclusively until they finish their mission. For efficiency, it can be executed by an administrator to gather these resources. Some running jobs may be suspended, and some waiting jobs may be held after the administrator negotiates with other users. When it comes to the reconfigurable network, it will be more complex because the resources include not only compute nodes but also Reconfigurable Links. A job may be unable to start without Reconfigurable Links. The performance may decrease without enough Reconfigurable Links. If the compute nodes and Reconfigurable Links allocation is complicated, it will be much more difficult for the administrator to gather these resources.

4.3.2 Large Jobs Share the Same Group

A frequently discussed question is if large jobs can share the same group. Assigning large jobs to the same group has to solve three problems: connectivity, performance, and complexity.

Firstly, the resource allocation policy has to guarantee topology connectivity for jobs. In a real supercomputing system, fragmented resources will occur as many jobs with different size start and finish. If a large job uses these fragmented resources regardless of the connectivity of topology, the nodes may not be able to communicate with each other. An example is shown in Figure 2a.

The more difficult problem is to guarantee the performance of jobs. If each large job uses dedicated Reconfigurable Links, the topology may not be reconfigured well, as shown in Figure 2b. Resource allocation policy has to guarantee appropriate and enough Reconfigurable Links are allocated for each large job. If the large jobs share the Reconfigurable Links in the same group, a Reconfigurable Links may need to be adjusted before a new job starts while the other running jobs may still use this link. Thus, some mechanism to clear Reconfigurable Links before they are disconnected must be designed to prevent reconfiguration from interfering with running jobs, which increases the reconfiguration time and complexity.

Suppose a new resource allocation policy can solve the connectivity and performance problems (maybe co-design with topology reconfiguration algorithm and routing schemes). In that case, the policy may be too complicated. It will be more difficult for the administrator to maintain the supercomputing system as claimed in Section 4.3.1. To the best of our knowledge, we do not find any solution that allows large jobs to share the same groups while guaranteeing performance and keeping the policy simple enough.

GRAP provides the best network performance for each job and solves the connectivity, performance, and complexity issues. Since each group is connected to all the OCSes, the groups and Reconfigurable Links allocated for each large job can form a smaller reconfigurable Dragonfly topology, which provides connectivity and performance for jobs. A large job can reconfigure the topology as it owns the whole system and won't interfere with another job. Last, GRAP is simple enough.

4.3.3 Complexity

The complexity of GRAP is negligible. GRAP needs O(N) space to maintain the number of idle, used and failure nodes in each group. The time complexity of GRAP is O(NlogN) deal to its sort procedures where N refers to the number of groups.

4.3.4 Adopting GRAP in Other Topologies

GRAP cannot be applied to traditional Dragonfly topology because it does not have Reconfigurable Links. Applying GRAP directly will cause link contention in global links. Previous works have shown that adaptive routing[27, 55] coupled with random/round-robin job placement[8, 9, 25] is a good solution for traditional Dragonfly topology. As for Fat-tree topology, a similar design[50] has been proposed. It also classifies the jobs according to their size to avoid inter-job interference. It's better to use their design in the Fat-tree topology. Therefore, GRAP is a resource allocation policy specially designed for reconfigurable Dragonfly topology.

5 EVALUATION

We designed a two-step experiment to evaluate the performance of HPC workloads in RDN-GRAP. In the first step, we evaluate the three resource allocation modes with corresponding traffic patterns under different job sizes. In the second step, we use the speedup results from the first step to evaluate a real HPC workload using Slurm Simulator with GRAP implemented in it.

5.1 Methodology

5.1.1 Simulators

We use two different simulators in each step. In the first step, we use the CODES (Co-Design of Multilayer Exascale Storage Architectures) network simulation toolkit[45]. CODES is a high-performance parallel discrete-event simulation framework targeting large-scale networking for HPC environments. CODES employs the Rensselaer Optimistic Simulator System (ROSS) [5, 10], a high-performance, parallel discrete-event simulator, to ensure massive simulations run accurately. CODES has already demonstrated its accuracy for various routing strategies on Dragonfly networks[44]. CODES has been used to evaluate the performance of MPI collective operations[17, 21], traffic characteristics[63], job interference[26, 66] in traditional Dragonfly topology.

In the second step, we use the Slurm Simulator from [54]. Simple Linux Utility for Resource Management (Slurm)[67] is an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system adopted in many supercomputer systems. And the Slurm Simulator is based on a real Slurm instance with modifications to allow the simulation of historical jobs and to improve the simulation speed. It has been used to study various schedule and resource allocation policy[47, 53, 56, 62]. The default resource allocation policy in Slurm for Dragonfly topology uses round-robin and load balance to avoid the hot spots in the network. In this work, we implement GRAP in Slurm Simulator to evaluate the performance of real HPC workload and compare it with its default resource allocation policy.

5.1.2 Network Simulation

We construct an RDN with 48-port EPSes and 288-port OCSes; each EPS connects to 12 compute nodes; each group contains 24 switches. There are 288 groups in the network, i.e., a total of 82944 nodes. In the first step simulation, the bandwidth is set to 12GB/s, and the latencies between two nodes are about 1.57us (1 hop in the same switch), 1.87us (2 hops in the same group), 2.07us (2 hops in different groups), 2.37us (3 hops), 2.67us (4 hops), which are close to performance evaluated on Tianhe-2A[34]. The congestion control developed by [39] and TAGO routing[57] are used in our simulation. As for the TDN used for comparison, the OCSes are removed, and the Progressive Adaptive Routing (PAR)[27] is used.

In addition, we introduce random network noise into the first step simulation. We set all other nodes send packets to each other randomly at 26% of its full speed, which is the same as [7, 17, 21, 30]. In the RDN, the noise traffic is limited within groups, according to GRAP.

5.1.3 Traffic Patterns

In the first step, we evaluate the performance of three traffic patterns: 3D Stencil Pattern, Neighboring Pattern, and All-to-All Pattern. These traffic patterns adopt three allocation modes defined in Section 5.2. The 3D Stencil Pattern adopts Balance Mode. It uses all the nodes in each allocated group. Each process communicates with 6 neighboring processes. The Neighboring Pattern adopts Custom Mode. It uses 288 or 216 nodes in each allocated group. Each process communicates with 4 processes within the same group and 4 processes in 4 neighboring groups. The All-to-All Pattern adopts Adaptive Mode and uses 256 nodes in each group (not fully used for comparing with Balance Mode). It will execute MPI_Alltoall optimized for Dragonfly topology by [21] several times, which can be seen as uniform traffic. As for the TDN used for comparison, random mapping is used for all these traffic patterns according to [8, 9, 25].

We would like to emphasize that this paper adopts many best practices from previous work, including routing schemes and task mapping in Dragonfly topology, and reconfiguration schemes in reconfigurable network. By no means is it intended to give complete details and validate these schemes since this would be well beyond the scope of this paper.

5.1.4 HPC Workload

In the second step, we use the public HPC workload history from the Mira supercomputing system[14, 19]. We simulate 12 workloads from the 12 months in 2019. There are 51946 jobs in total. Because Mira only contains 49152 nodes, we enlarge the workload by doubling the number of nodes requested by each job. Then, if the doubled job size exceeds 82944, we will reduce the job size to 82944. This public HPC workload history also contains time consumed in MPI for some jobs, but not all jobs (the reasons can be found in [14]). We calculate the average MPI time percentage for different sizes of jobs in 2019. We assume RDN-GRAP can only accelerate the MPI time.

In the Reconfigurable Dragonfly topology, we expect most users of large jobs to use RDN properly. Thus we assume that, for the large jobs with the size of 576~65536, 40% of them use Minimal Balance Mode, 40% of them use Adaptive Mode, 10% of them use Custom Balance Mode, and 10% of them use Custom Mode. As for the large jobs with sizes above 65536 or less than 576, we assume they use Minimal Balanced Mode. The runtime of a job accelerated by RDN is calculated by its size and allocation mode:

$$Acc\_Time = Job\_Time - MPI\_Time \times Speedup(Mode, Job\_Size)$$

We use the accelerated-enlarged workload as input for Slurm Simulator with GRAP implemented in it. As for comparison, we use the enlarged workload as input for Slurm Simulator, which uses the default resource allocation policy for Dragonfly in Slurm.

5.2 Allocation Mode Evaluation

Figure 4 shows the performance of different traffic patterns with corresponding allocation modes. The baseline is the performance of TDN with network noise. We analyze their common characteristics first. In the TDN, the performance is affected by network noise. About 10% of performance is lost in small job sizes. However, in RDN-GRAP, the performance is not affected by network noise because large jobs are isolated by GRAP and small jobs hardly affect large jobs. By the way, the results of RDN-GRAP with noise is not shown in Figure 4c. The job is not affected by network noise

(a) All-to-All Pattern using Adaptive Mode

(b) Neighboring Pattern Hops

(c) 3D Stencil Pattern using Balance Mode

(d) Neighboring Pattern using Custom Mode

Figure 4: Allocation Mode Evaluation Results. RDN-GRAP achieves inter-job interference-free, lower latency, and higher bandwidth. It works better when the job uses fewer groups and the hot spots are more concentrated.

because it uses all the nodes in the groups. There is no common link with other jobs after network reconfiguration.

Another common characteristic is that RDN-GRAP achieves higher bandwidth and lower latency. In RDN-GRAP, each group only needs to connect groups owned by the same large job, and the topology is configured according to the application's traffic pattern. Thus, the bandwidth is increased. As for latency, large jobs can take advantage of both intra-group communication performance provided by GRAP and multiple paths provided by RDN. In TDN, most packets have to cross groups through global links and even take a detour to avoid congestion when network noise occurs. Figure 4b shows the hops statistics for Neighboring Pattern using 8 groups with Custom Mode as an example. The average hops in RDN-GRAP reduces 47% and is not affected by network noise. All the packets in RDN take no more than 3 hops, while most packets in TDN take more than 3 hops.

The performance of All-to-All Pattern using Adaptive Mode is shown in Figure 4a. RDN-GRAP shows more speedup when a small number of groups is used. It shows a 32% speedup compared to TDN with network noise when using two groups. This is because it reconfigures the topology to a smaller Dragonfly topology. The number of direct global links between two groups is much more than the original topology when using a small number of groups. However, the speedup decreases as the number of used groups increase, and the performance becomes the same when using all the groups.

As for the 3D Stencil Pattern and Neighboring Pattern (Figure 4c and 4d), their relative speedup is stable as the size increase. This is because each group only communicates with a limited number of other groups. RDN-GRAP works much better in such traffic patterns by reconfiguring the topology. 3D Stencil Pattern accelerates about 30% and Neighboring Patter accelerates about 16%. The speedup of Neighboring Pattern is lower than the 3D Stencil Pattern because each group in Neighboring Pattern communicates with more other groups. Therefore, RDN-GRAP works better when the job uses fewer groups and the hot spots are more concentrated.

In addition, we use All-to-All Pattern to compare the Adaptive Mode and Balance Mode as shown in Figure 4a. In each group, Balance Mode uses 256 nodes, and Adaptive Mode uses about 256 nodes. The results show that there is almost no difference in performance. Thus, the Adaptive Mode is more suitable for this traffic pattern because it provides more flexibility in resource allocation and achieves the same performance.

5.3 Workload Evaluation

According to the speedup results from the previous subsection and the workload composition assumption in Section 5.1.4, we set the input for RDN-GRAP Slurm Simulator. The average speedup of jobs

(a) Average Job Wait Time Decrease Percentage in each month of 2019

(b) System Utilization in each month of 2019

Figure 5: Workload Evaluation. The speedup produced by RDN-GRAP for jobs covers its drawbacks caused by the restrictions. RDN-GRAP reduces job wait time by 26% on average. Lower system utilization in RDN-GRAP indicates that more jobs can be submitted to RDN-GRAP, and users should set more appropriate sizes for jobs.

is 9% ~11% for each month after considering the real communication time of applications. The results are shown in Figure 5.

Figure 5a shows the average job wait time decrease percentage in RDN-GRAP compared to TDN. The job wait time is calculated by job start time minus job submit time. It reduces 9%~52% and 26% on average. Although the various decrement in job wait time shows that the improvement of RDN-GRAP depends on specific workloads, none of the results show longer job wait time. It means the speedup produced by RDN-GRAP for jobs can cover its drawbacks caused by restrictions in GRAP.

Intuitively, it may be weird that job wait time decreases much more than the speedup of jobs, but the queuing theory can explain it. Note that we maintain the job submission time in the simulation instead of submitting all the jobs at the start of the simulation. For example, two jobs are submitted at the beginning and the 8th minute. The first job runs for 10 minutes, and the second job has to wait for the first job to finish. The second job needs to wait 2 minutes. If the first job accelerates 10%, the second job only needs to wait 1 minute, and its wait time decreases 50%.

The utilization of the whole system in TDN and RND-GRAP is shown in Figure 5b. The utilization is calculated by counting the number of nodes allocated every minute. The box and whisker plots show the distribution over the entire simulation. The system utilization of RDN-GRAP is lower than TDN. There are two reasons: firstly, RDN-GRAP accelerates the jobs, and some nodes become idle after the accelerated jobs finish; secondly, GRAP restricts resource allocation, which leads to some resources that violate GRAP's rules cannot be allocated to waiting jobs. It indicates that more jobs can be submitted to RDN-GRAP, and the system administrator should guide users to set more appropriate sizes for jobs to reduce waiting time and improve system utilization. For example, the system administrator can add the number of idle nodes in each group as prompt information for small job into sinfo command for Slurm. As for large job, the prompt information should be the number of groups that do not contain large jobs and the number of idle nodes in these groups.

By the way, the low system utilization in September may be caused by system maintenance. No job has been submitted for several days in September.

6 RELATED WORKS

Common OCSes are Micro-Electro-Mechanical Systems (MEMS) based OCS and Arrayed Wavelength Grating Routers (AWGR). MEMS-based OCS redirects the photonic signals from each input fiber to one of the output fibers by tiltable mirrors[48]. AWGR leverages wavelength routing, silicon photonic microring resonators' reconfigurable add/drop filtering capabilities, and a multiwavelength switch[35].

In this work, we assume MEMS-based OCS is used for the reconfigurable network for three reasons. First, its hundred milliseconds reconfiguration time is fast enough for GRAP, even though AWGR can achieve microseconds reconfiguration time. Second, MEMS-based OCS is compatible with the current EPS and not limited by link bandwidth because it physically reflects the light beam from in-port to out-port. Even if the network bandwidth is upgraded, the MEMS-based OCS is still compatible[52]. Third, it is cheaper and easier to implement more ports. MEMS-based OCS with 320 ports is currently available[1]; meanwhile, 512 and 1024 ports MEMS-based OCSes are on their way. In this paper, OCS refers to MEMS-based OCS by default unless otherwise specified.

OCSes usually construct network topology with EPSes to combine both of their advantages. The OCS has more ports and lower latency, and is power-saving, while the EPS is more flexible because of the packet switch. Many different architectures have been proposed for the reconfigurable network. They can be categorized into two classes[59]: ToR-reconfigurable network (TRN) and pod-reconfigurable network (PRN). In TRN, OCSes are placed among top-of-rank (ToR) EPSes, such as 3D-Hyper-FleX-LION[35], which add OCSes among ToR EPSes for each dimension in 3D HyperX topology. In PRN, OCSes are placed among pods of racks, such as FlexFly[64], which add OCSes among groups in Dragonfly topology. In this work, RDN is similar to FlexFly and belongs to PRN.

The reconfigurable network has been adopted in Google's data center. Their research[52] shows that their reconfigurable network delivered 5x higher speed and capacity, 30% reduction in capex, and 41% reduction in power. They use MEMS-based OCSes and centralized Software-Defined Networking (SDN) control for traffic engineering, and automated network operations for incremental

capacity delivery and topology engineering. Reconfigurable networks in the data center have been widely discussed. More works can be found in [3, 12, 20, 23, 24, 36, 40, 41, 51, 61].

7 CONCLUSION

In this work, we aim to design a resource allocation policy for Reconfigurable Dragonfly Network in HPC. To let the resource allocation policy can be applied to real supercomputing systems, we proposed three design principles first: reconfigurable network should be reconfiguration interference-free, guarantee connectivity and performance for each job, and satisfy varied resource requests.

Based on these design principles, we proposed GRAP. GRAP first classifies the jobs into small and large jobs. Small jobs are allocated nodes within a group and do not use Reconfigurable Links. A large job will be allocated with some groups that do not contain any other large jobs and all the Reconfigurable Links attached to these groups. GRAP further provides three resource allocation modes for large job users to let applications achieve the best performance. GRAP advocate advanced users should understand the basic concept of Reconfigurable Dragonfly Network, configure the input of their jobs, and choose an appropriate allocation mode in GRAP to fully utilize the performance provided by RDN and accelerate their jobs.

The key design of GRAP is job classification and restriction. It achieves interference-free reconfiguration and limits inter-job interference through a concise and understandable design. It also makes GRAP compatible with many previous works, including routing schemes and topology reconfiguration algorithms for reconfigurable networks. GRAP enables them to be adopted into a real HPC system. In addition, future works can be based on GRAP and assume that a smaller RDN is allocated for each large job and large jobs won't interfere with each other. It dramatically reduces the complexity of corresponding algorithm design, such as topology reconfiguration algorithms, routing algorithms, and collective communication algorithms.

The evaluation results show that RDN-GRAP achieves inter-job interference-free, lower latency, higher bandwidth, and lower job wait time compared to TDN. It works better when the job uses fewer groups and the hot spots are more concentrated. The speedup produced by RDN-GRAP for jobs covers its drawbacks caused by the restrictions and reduces job wait time by 26% on average. Although RDN-GRAP causes lower system utilization, it indicates that more jobs can be submitted to RDN-GRAP, and system administrators should guide users to set more appropriate sizes for jobs to reduce waiting time and improve system utilization.

ACKNOWLEDGMENTS

Thanks to Hua Huang and Wenkai Shao who gave many valuable comments on this work. And thanks to everyone who has contributed to this work in any way. This work is supported by the National Key R&D Program of China under Grant NO. 2022YFB4500304, the Major Program of Guangdong Basic and Applied Research: 2019B030302002, National Natural Science Foundation of China (NSFC): 62272499, Guangdong Province Special Support Program for Cultivating High-Level Talents: 2021TQ06X160, and Excellent Youth Foundation of Hunan Province under Grant No. 2021JJ10050. Yutong Lu is the corresponding author of this paper.

REFERENCES

[1] 2022. Photonic Optical Circuit Switching | CALIENT Technologies. https://www.calient.net/
[2] 2023. Home | Community Earth System Model. https://www.cesm.ucar.edu/
[3] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Benn Thomsen, Kai Shi, and Hugh Williams. 2020. Sirius: A Flat Datacenter Network with Nanosecond Optical Switching. https://www.microsoft.com/en-us/research/publication/sirius-a-flat-datacenter-network-with-nanosecond-optical-switching/
[4] K.J. Barker, A. Benner, R. Hoare, A. Hoisie, A.K. Jones, D.K. Kerbyson, D. Li, R. Melhem, R. Rajamony, E. Schenfeld, S. Shao, C. Stunkel, and P. Walker. 2005. On the Feasibility of Optical Circuit Switching for High Performance Computing Systems. In *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*. 16–16. https://doi.org/10.1109/SC.2005.48
[5] Peter D. Barnes, Christopher D. Carothers, David R. Jefferson, and Justin M. LaPre. 2013. Warp speed: executing time warp on 1,966,080 cores. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM PADS '13)*. Association for Computing Machinery, New York, NY, USA, 327–336. https://doi.org/10.1145/2486092.2486134
[6] Tal Ben-Nun and Torsten Hoefler. 2019. Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis. *Comput. Surveys* 52, 4 (Aug. 2019), 65:1–65:43. https://doi.org/10.1145/3320060
[7] Maciej Besta and Torsten Hoefler. 2014. Slim fly: a cost effective low-diameter network topology. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*. IEEE Press, New Orleans, Louisana, 348–359. https://doi.org/10.1109/SC.2014.34
[8] Abhinav Bhatele, William D. Gropp, Nikhil Jain, and Laxmikant V. Kale. 2011. Avoiding hot-spots on two-level direct networks. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11. https://doi.org/10.1145/2063384.2063486 ISSN: 2167-4337.
[9] J. Brandt, K. Devine, A. Gentile, and K. Pedretti. 2014. Demonstrating improved application performance using dynamic monitoring and task mapping. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. 408–415. https://doi.org/10.1109/CLUSTER.2014.6968670 ISSN: 2168-9253.
[10] Christopher D. Carothers, David Bauer, and Shawn Pearce. 2000. ROSS: a high-performance, low memory, modular time warp system. In *Proceedings of the fourteenth workshop on Parallel and distributed simulation (PADS '00)*. IEEE Computer Society, USA, 53–60. https://ieeexplore.ieee.org/document/847144
[11] Swiss National Supercomputing Centre(CSCS). 2018. Factsheet: "Piz Daint", one of the most powerful supercomputers in the world. https://www.cscs.ch/publications/news/2017/factsheetpizdaintoneofthemostpowerfulsupercomputersintheworld/
[12] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. 2014. OSA: An Optical Switching Architecture for Data Center Networks With Unprecedented Flexibility. *IEEE/ACM Transactions on Networking* 22, 2 (April 2014), 498–511. https://doi.org/10.1109/TNET.2013.2253120 Conference Name: IEEE/ACM Transactions on Networking.
[13] Xiaoliang Chen, Roberto Proietti, Marjan Fariborz, Che-Yu Liu, and S. J. Ben Yoo. 2021. Machine-learning-aided cognitive reconfiguration for flexible-bandwidth HPC and data center networks [Invited]. *Journal of Optical Communications and Networking* 13, 6 (June 2021), C10–C20. https://doi.org/10.1364/JOCN.412360 Conference Name: Journal of Optical Communications and Networking.
[14] Sudheer Chunduri, Scott Parker, Pavan Balaji, Kevin Harms, and Kalyan Kumaran. 2018. Characterization of MPI Usage on a Production Supercomputer. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Dallas, TX, USA, 386–400. https://doi.org/10.1109/SC.2018.00033
[15] Pedro Costa. 2018. A FFT-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows. *Computers & Mathematics with Applications* 76, 8 (Oct. 2018), 1853–1862. https://doi.org/10.1016/j.camwa.2018.07.034
[16] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler. 2020. An In-Depth Analysis of the Slingshot Interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Atlanta, GA, USA, 1–14. https://doi.org/10.1109/SC41405.2020.00039
[17] Matthieu Dorier, Misbah Mubarak, Rob Ross, Jianping Kelvin Li, Christopher D. Carothers, and Kwa-Liu Ma. 2016. Evaluation of Topology-Aware Broadcast Algorithms for Dragonfly Networks. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. 40–49. https://doi.org/10.1109/CLUSTER.2016.26 ISSN: 2168-9253.
[18] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. 2012. Cray Cascade: A scalable HPC system based on a Dragonfly network. In *2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '12)*. IEEE, Salt Lake City, UT, 1–9. https://doi.org/10.1109/SC.2012.39

[19] Argonne Leadership Computing Facility(ALCF). 2022. ALCF Public Data. https://reports.alcf.anl.gov/data/index.html

[20] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2010. Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM Computer Communication Review* 40, 4 (Aug. 2010), 339–350. https://doi.org/10.1145/1851275.1851223

[21] Guangnan Feng, Dezun Dong, and Yutong Lu. 2022. Optimized MPI collective algorithms for dragonfly topology. In *Proceedings of the 36th ACM International Conference on Supercomputing (ICS '22)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3524059.3532380

[22] Pablo Fuentes, Enrique Vallejo, Cristóbal Camarero, Ramón Beivide, and Mateo Valero. 2015. Throughput Unfairness in Dragonfly Networks under Realistic Traffic Patterns. In *2015 IEEE International Conference on Cluster Computing*. 801–808. https://doi.org/10.1109/CLUSTER.2015.136 ISSN: 2168-9253.

[23] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. 2016. ProjecToR: Agile Reconfigurable Data Center Interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 216–229. https://doi.org/10.1145/2934872.2934911

[24] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R. Das, Jon P. Longtin, Himanshu Shah, and Ashish Tanwer. 2014. FireFly: a reconfigurable wireless data center fabric using free-space optics. *ACM SIGCOMM Computer Communication Review* 44, 4 (Aug. 2014), 319–330. https://doi.org/10.1145/2740070.2626328

[25] Nikhil Jain, Abhinav Bhatele, Xiang Ni, Nicholas J. Wright, and Laxmikant V. Kale. 2014. Maximizing Throughput on a Dragonfly Network. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*. IEEE, New Orleans, LA, USA, 336–347. https://doi.org/10.1109/SC.2014.33

[26] Nikhil Jain, Abhinav Bhatele, Sam White, Todd Gamblin, and Laxmikant V. Kale. 2016. Evaluating HPC Networks via Simulation of Parallel Workloads. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16)*. 154–165. https://doi.org/10.1109/SC.2016.13 ISSN: 2167-4337.

[27] Nan Jiang, John Kim, and William J. Dally. 2009. Indirect adaptive routing on large scale interconnection networks. In *Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09)*. Association for Computing Machinery, New York, NY, USA, 220–231. https://doi.org/10.1145/1555754.1555783

[28] Tyson Jones, Anna Brown, Ian Bush, and Simon C. Benjamin. 2019. QuEST and High Performance Simulation of Quantum Computers. *Scientific Reports* 9, 1 (July 2019), 10736. https://doi.org/10.1038/s41598-019-47174-9 Number: 1 Publisher: Nature Publishing Group.

[29] Norman P Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings: Industrial Product. In *The 50th Annual International Symposium on Computer Architecture (ISCA '23)*. ACM, New York, NY, USA, June 17-21, 2023, Orlando, FL, USA, 15 pages. https://doi.org/10.1145/3579371.3589350

[30] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *2008 International Symposium on Computer Architecture (ISCA '08)*. 77–88. https://doi.org/10.1109/ISCA.2008.19 ISSN: 1063-6897.

[31] Los Alamos National Laboratory(LANL)LANL. 2020. Trinity: Advanced Technology System. https://www.lanl.gov/projects/trinity/

[32] Sylvain Laizet and Ning Li. 2010. Incompact3d: a powerful tool to tackle turbulence problems with up to hundreds of thousands computational cores. (Nov. 2010). https://www.researchgate.net/publication/253825870_Incompact3d_a_powerful_tool_to_tackle_turbulence_problems_with_up_to_hundreds_of_thousands_computational_cores

[33] Jianping Kelvin Li, Misbah Mubarak, Robert B. Ross, Christopher D. Carothers, and Kwan-Liu Ma. 2017. Visual Analytics Techniques for Exploring the Design Space of Large-Scale High-Radix Networks. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Honolulu, HI, USA, 193–203. https://doi.org/10.1109/CLUSTER.2017.26

[34] Xiang-Ke Liao, Zheng-Bin Pang, Ke-Fei Wang, Yu-Tong Lu, Min Xie, Jun Xia, De-Zun Dong, and Guang Suo. 2015. High Performance Interconnect Network for Tianhe System. *Journal of Computer Science and Technology* 30, 2 (March 2015), 259–272. https://doi.org/10.1007/s11390-015-1520-7

[35] Gengchen Liu, Roberto Proietti, Marjan Fariborz, Pouya Fotouhi, Xian Xiao, and S. J. Ben Yoo. 2020. Architecture and performance studies of 3D-Hyper-FleX-LION for reconfigurable all-to-all HPC networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*. IEEE Press, Atlanta, Georgia, 1–16. https://dl.acm.org/doi/10.5555/3433701.3433735

[36] He Liu, Matthew K. Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M. Voelker, David G. Andersen, Michael Kaminsky, George Porter, and Alex C. Snoeren. 2015. Scheduling techniques for hybrid circuit/packet networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '15)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/2716281.2836126

[37] Hao Lu, Michael Matheson, Vladyslav Oles, Austin Ellis, Wayne Joubert, and Feiyi Wang. 2022. Climbing the Summit and Pushing the Frontier of Mixed Precision Benchmarks at Extreme Scale. IEEE Computer Society, 1123–1137. https://www.computer.org/csdl/proceedings-article/sc/2022/544400b123/1I0bTcFWPzW ISSN: 2167-4337.

[38] Pavlos Maniotis, Nicolas Dupuis, Laurent Schares, Daniel M. Kuchta, Marc A. Taubenblatt, and Benjamin G. Lee. 2020. Intra-node high-performance computing network architecture with nanosecond-scale photonic switches [Invited]. *Journal of Optical Communications and Networking* 12, 12 (Dec. 2020), 367–377. https://doi.org/10.1364/JOCN.399925 Conference Name: Journal of Optical Communications and Networking.

[39] Neil McGlohon, Christopher D. Carothers, K. Scott Hemmert, Michael Levenhagen, Kevin A. Brown, Sudheer Chunduri, and Robert B. Ross. 2021. Exploration of Congestion Control Techniques on Dragonfly-class HPC Networks Through Simulation. In *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 40–50. https://doi.org/10.1109/PMBS54543.2021.00010

[40] William M. Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C. Snoeren, and George Porter. 2020. Expanding across time to deliver bandwidth efficiency and low latency. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation (NSDI'20)*. USENIX Association, USA, 1–18. https://www.usenix.org/conference/nsdi20/presentation/mellette

[41] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. 2017. RotorNet: A Scalable, Low-complexity, Optical Datacenter Network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 267–280. https://doi.org/10.1145/3098822.3098838

[42] George Michelogiannakis, Yiwen Shen, Min Yee Teh, Xiang Meng, Benjamin Aivazi, Taylor Groves, John Shalf, Madeleine Glick, Manya Ghobadi, Larry Dennison, and Keren Bergman. 2019. Bandwidth steering in HPC using silicon nanophotonics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19)*. Association for Computing Machinery, New York, NY, USA, 1–25. https://doi.org/10.1145/3295500.3356145

[43] Cyriel Minkenberg, German Rodriguez, Bogdan Prisacari, Laurent Schares, Philip Heidelberger, Dong Chen, and Craig Stunkel. 2016. Performance benefits of optical circuit switches for large-scale dragonfly networks. In *2016 Optical Fiber Communications Conference and Exhibition (OFC)*. 1–3. https://ieeexplore.ieee.org/document/7537792

[44] Misbah Mubarak, Christopher D. Carothers, Robert Ross, and Philip Carns. 2012. Modeling a Million-Node Dragonfly Network Using Massively Parallel Discrete-Event Simulation. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, Salt Lake City, UT, 366–376. https://doi.org/10.1109/SC.Companion.2012.56

[45] Misbah Mubarak, Christopher D. Carothers, Robert B. Ross, and Philip Carns. 2017. Enabling Parallel Simulation of Large-Scale HPC Network Systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (Jan. 2017), 87–100. https://doi.org/10.1109/TPDS.2016.2543725

[46] NERSC. 2021. Perlmutter System Details. https://docs.nersc.gov/systems/perlmutter/system_details/

[47] Tatsuyoshi Ohmura, Yoichi Shimomura, Ryusuke Egawa, and Hiroyuki Takizawa. 2023. Toward Building a Digital Twin of Job Scheduling and Power Management on an HPC System. In *Job Scheduling Strategies for Parallel Processing (Lecture Notes in Computer Science)*, Dalibor Klusáček, Corbalán Julita, and Gonzalo P. Rodrigo (Eds.). Springer Nature Switzerland, Cham, 47–67. https://doi.org/10.1007/978-3-031-22698-4_3

[48] Flavio Pardo, Vladimir A. Aksyuk, Susanne Arney, H. Bair, Nagesh R. Basavanhally, David J. Bishop, Gregory R. Bogart, Cristian A. Bolle, J. E. Bower, Dustin Carr, H. B. Chan, Raymond A. Cirelli, E. Ferry, Robert E. Frahm, Arman Gasparyan, John V. Gates, C. Randy Giles, L. Gomez, Suresh Goyal, Dennis S. Greywall, Martin Haueis, R. C. Keller, Jungsang Kim, Fred P. Klemens, Paul R. Kolodner, Avi Kornblit, T. Kroupenkine, Warren Y.-C. Lai, Victor Lifton, Jian Liu, Yee L. Low, William M. Mansfield, Dan Marom, John F. Miner, David T. Neilson, Mark A. Paczkowski, C. S. Pai, A. G. Ramirez, David A. Ramsey, S. Rogers, Roland Ryf, Ronald E. Scotti, Herbert R. Shea, M. E. Simon, H. T. Soh, Hong Tang, J. A. Taylor, K. Teffeau, Joseph Vuillemin, and J. Weld. 2003. Optical MEMS devices for telecom systems. In *Smart Sensors, Actuators, and MEMS*, Vol. 5116. SPIE, 435–444. https://doi.org/10.1117/12.499075

[49] Tirthak Patel, Zhengchun Liu, Raj Kettimuthu, Paul Rich, William Allcock, and Devesh Tiwari. 2020. Job characteristics on large-scale systems: long-term analysis, quantification, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*. IEEE Press, Atlanta, Georgia, 1–17. https://dl.acm.org/doi/10.5555/3433701.3433812

[50] Samuel D. Pollard, Nikhil Jain, Stephen Herbein, and Abhinav Bhatele. 2018. Evaluation of an Interference-free Node Allocation Policy on Fat-tree Clusters. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Dallas, TX, USA, 333–345. https://doi.org/10.1109/SC.2018.00029

[51] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2013. Integrating microsecond circuit switching into the data center. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 447–458. https://doi.org/10.1145/2486001.2486007

[52] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. 2022. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 66–85. https://doi.org/10.1145/3544216.3544265

[53] Jessi Christa Rubio, Aira Villapando, Christian Matira, and Jeffrey Aborot. 2020. Correcting Job Walltime in a Resource-Constrained Environment. In *Supercomputing Frontiers (Lecture Notes in Computer Science)*, Dhabaleswar K. Panda (Ed.). Springer International Publishing, Cham, 118–137. https://doi.org/10.1007/978-3-030-48842-0_8

[54] Nikolay A. Simakov, Martins D. Innus, Matthew D. Jones, Robert L. DeLeon, Joseph P. White, Steven M. Gallo, Abani K. Patra, and Thomas R. Furlani. 2018. A Slurm Simulator: Implementation and Parametric Analysis. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*. Springer, Cham, 197–217. https://doi.org/10.1007/978-3-319-72971-8_10

[55] Arjun Singh. 2005. *LOAD-BALANCED ROUTING IN INTERCONNECTION NETWORKS*. Ph. D. Dissertation. Stanford University.

[56] Mohammed Tanash, Brandon Dunn, Daniel Andresen, William Hsu, Huichen Yang, and Adedolapo Okanlawon. 2019. Improving HPC System Performance by Predicting Job Resources via Supervised Machine Learning. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning) (PEARC '19)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3332186.3333041

[57] Min Yee Teh, Yu-Han Hung, George Michelogiannakis, Shijia Yan, Madeleine Glick, John Shalf, and Keren Bergman. 2020. TAGO: Rethinking Routing Design in High Performance Reconfigurable Networks. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–16. https://doi.org/10.1109/SC41405.2020.00029

[58] Min Yee Teh, Zhenguo Wu, and K. Bergman. 2020. Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering. *IEEE/OSA Journal of Optical Communications and Networking* (2020). https://doi.org/10.1364/JOCN.379487

[59] Min Yee Teh, Zhenguo Wu, Madeleine Glick, Sebastien Rumley, Manya Ghobadi, and Keren Bergman. 2022. Performance trade-offs in reconfigurable networks for HPC. *Journal of Optical Communications and Networking* 14, 6 (June 2022), 454–468. https://doi.org/10.1364/JOCN.451760 Conference Name: Journal of Optical Communications and Networking.

[60] O. Tuncer, Yijia Zhang, V. Leung, and A. Coskun. 2017. *Task mapping on a dragonfly supercomputer*. Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States). https://www.semanticscholar.org/paper/Task-mapping-on-a-dragonfly-supercomputer-Tuncer-Zhang/ac5416c4d080fabf5b983f86a5497f487389e9a9

[61] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. 2010. c-Through: part-time optics in data centers. *ACM SIGCOMM Computer Communication Review* 40, 4 (Aug. 2010), 327–338. https://doi.org/10.1145/1851275.1851222

[62] Hao Wang, Yi-Qin Dai, Jie Yu, and Yong Dong. 2021. Predicting running time of aerodynamic jobs in HPC system by combining supervised and unsupervised learning method. *Advances in Aerodynamics* 3, 1 (Aug. 2021), 22. https://doi.org/10.1186/s42774-021-00077-8

[63] Xin Wang, Misbah Mubarak, Xu Yang, Robert B. Ross, and Zhiling Lan. 2018. Trade-Off Study of Localizing Communication and Balancing Network Traffic on a Dragonfly System. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, Vancouver, BC, 1113–1122. https://doi.org/10.1109/IPDPS.2018.00120

[64] Ke Wen, Payman Samadi, Sébastien Rumley, Christine P. Chen, Yiwen Shen, Meisam Bahadori, Keren Bergman, and Jeremiah Wilke. 2016. Flexfly: Enabling a Reconfigurable Dragonfly through Silicon Photonics. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 166–177. https://doi.org/10.1109/SC.2016.14 ISSN: 2167-4337.

[65] Jiabin Xie, Jianchao He, Yun Bao, and Xi Chen. 2021. A Low-Communication-Overhead Parallel DNS Method for the 3D Incompressible Wall Turbulence. *International Journal of Computational Fluid Dynamics* 35, 6 (July 2021), 413–432. https://doi.org/10.1080/10618562.2021.1971202 Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/10618562.2021.1971202.

[66] Xu Yang, John Jenkins, Misbah Mubarak, Robert B. Ross, and Zhiling Lan. 2016. Watch Out for the Bully! Job Interference Study on Dragonfly Network. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Salt Lake City, UT, USA, 750–760. https://doi.org/10.1109/SC.2016.63

[67] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing (Lecture Notes in Computer Science)*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer, Berlin, Heidelberg, 44–60. https://doi.org/10.1007/10968987_3

[68] Felix Zahn and Holger Fröning. 2020. On Network Locality in MPI-Based HPC Applications. In *49th International Conference on Parallel Processing - ICPP (ICPP '20)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3404397.3404436

[69] Zuoqing Zhao, Bingli Guo, Yu Shang, and Shanguo Huang. 2020. Hierarchical and reconfigurable optical/electrical interconnection network for high-performance computing. *Journal of Optical Communications and Networking* 12, 3 (March 2020), 50–61. https://doi.org/10.1364/JOCN.377427 Conference Name: Journal of Optical Communications and Networking.