# Topology-aware Allocation Policies for Jellyfish

Jose A. Pascual, Javier Navaridas, Alejandro Erickson and Ian

## ABSTRACT

## 1. INTRODUCTION

Jellyfish topology [1] has been recently proposed as a high bandwidth and low latency interconnect for large scale data centers and HPC systems. In opposite to recenctly proposed server-centric datacenter networks (DCN), jellyfish is an indirect (switch-centric) network in which the servers are connected to the switches. This network is a *degree-bounded* random regular graph (RRG) among the top-of-rack (ToR) switches, in which all the nodes have the same degree, are bidirectional and are connected randomly. RRGs also provide other desiderable properties for an interconnect such as low diameter, high connectivity among nodes and easy incremental expansion.

As stated in the original paper [1], routing in jellyfish is a challenge. Although jellyfish provides high connectivity among switches, classical routing policies are not able to exploit the path diversity offered. In that work the authors evaluated two well-known routing policies, shortest path (SP) and Equal-cost Multi-path (ECMP), assesing that the use of the shortest paths does not provide enough path diversity to utilize the full capacity of the network. This issue was solved using the K-Shortest Path (KSP) [2] routing policy that uses more paths at the cost of being longer. Although KSP perfoms well compared to SP and ECMP, the author in [3] showed that jellyfish has several features that make it ineffective. In particular they stated the possibly large number of source-destinations (SD) pairs that will share the same K shortest-paths and the random number of short paths between each pair of switches.

These works have studied jellyfish both theoretically, puting bounds to topological properties, and empirically evaluating the performance of several communication patterns. However none of them have consider the natural scenario in which this topology could be used: data centers or HPC centers where many applications run concurrently. To the best of our knowledge, there is no work devoted to evaluate the performance of such applications in this topology. The assignemnt of resources to application has been widely studied in the context of HPC. Those works clearly differenciate three stages: selection of the application to be executed, allocation of the resources to that application and mapping of the taks that compose the application to the physical servers.

The objective of this work is the evaluation of existing allocation policies applied to jellyfish in multi-application scenarios. We will see how these policies do not perform well in jellyfish due to the random nature of this topology. Because of that, we will apply ideas taken from the HPC world and adapt them to the particularisms of jellyfish. In particular we want to define concepts of locality to jellyfish. In addition we will extend the definition of contiguity and convexity to jellyfish. These policies should improve the performance of applications executed on jellyfish. However, contiguity can severely affects system utilization. this drawback will also be studied.

The evaluation of the allocation policies has been carried out using extensive simulation with a representative mix of parallel application. Results show that tradicional and simple allocation strategies do not take advantadge of the characteristics of jellyfish. Moreover we will se how locality-aware policies can take advantadge of those properties improving the performance of applications running concurrently. In particulae locality and contiguous policies ..., We will also evaluate the drawbacks....

The rest of the paper is organized as follow. Section 2 introduces jellyfish and communication patterns used thorough the paper. We continue in Section 3 analyzing simple allocation policies for jellyfish introducing in Sections [**?**] and [**?**] the proposed policies. The results are presented and analyzed in Section [**?**] concluding in Section **??** with some conlucsions and future lines of work.

## 2. BACKGROUND

In this section we describe thoroughly the jellyfish topology and give some definitions and properties that will be used in the rest of the paper. Furthermore, we describe the four routing policies developed for jellyfish asessing the pros and cons of each one of them.

### 2.1 Jellyfish topology

Jellyfish is a network topology in which the switches are connected randomly. We denote jellyfish as RRG(N,k,r) where N is the number of switches, k is the number of ports of the switches and r is the number of ports used to connect to other switches.

### 2.2 Routing policies

The jellyfish topology provides high path diversity between any pair of servers. This is one of the characteristic that makes this topology so appealing to execute parallel workloads. As we have seen, there exists a lot of possible

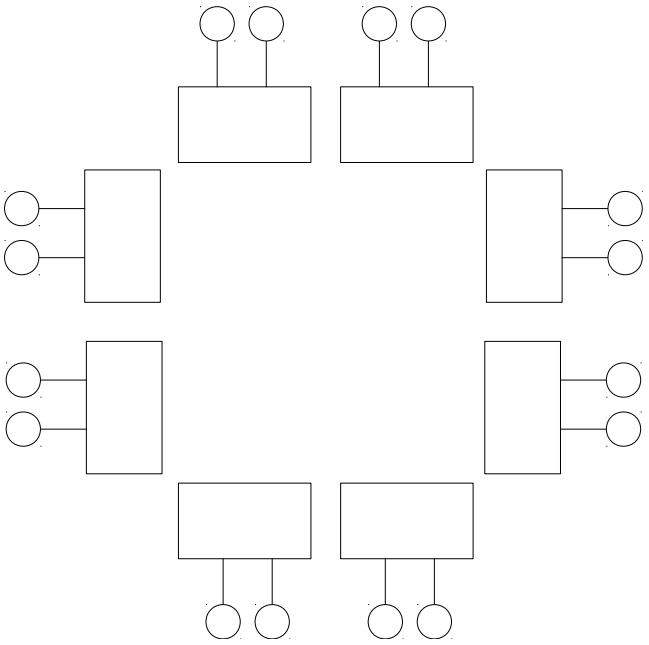


Figure 1

paths among physical servers to carry out the communications. This is the reason why the chosing of a routing policy is so important in jellyfish in order to avoid contention in network links. There exist only four routing policies proposed and evaluated for jellyfish. Next, we will analize these policies stating the improvements of one against the others.

- Shortest Path (SP):
- Equal-Cost Multi-Path (ECMP):
- K-Shortest Path:
- Limited Length Spread K-shortest Path (LLKSR):

## 2.3 Workloads and performance metric

- Nearest neighbor:
- Collective:
- : Permutation and shift pattern:

# 3. ALLOCATION IN JELLYFISH

When an application is submitted to be executed, the allocator is in charge of finding the appropiate set of resources to place it. There are many works that propose allocation policies for other topologies such as meshes, tori [4] [5], or fattrees [5] [6], but, to the best of our knowledge, not for jellyfish. Only in [3] the authors use some simple mono-application allocation policies without taking into account how these policies behave when multiple application run concurrently. In this section we describe and analize those policies adding a new version of one of them due to random structure of jellyfish.

In order to define the allocation policies we must first assign assign a node ordering to the servers and switches that compose the network. Due to the random structure of jellyfish, we first define order the switches and then, inside each switch, we order the server consecutively.

- Sequential: The tasks are assigned to the servers consecutively using the identity function.

$$\pi : T \rightarrow P\pi(t_i) = p_i \quad (1)$$

- Random: The tasks of the application are assigned to the servers uniformly at random.

$$\pi : T \rightarrow P\pi(t_i) = p_j where p_j is selected at random \quad (2)$$

- Random$^*$: This is a special version of the previouly presented policies. In this case we select a switch uniformy at random and then we map (k-r) tasks to the servers direcctly connected to it.

$$\pi : T \rightarrow P\pi(t_i) = p_j where p_j is selected at random \quad (3)$$

LEMMA 1.

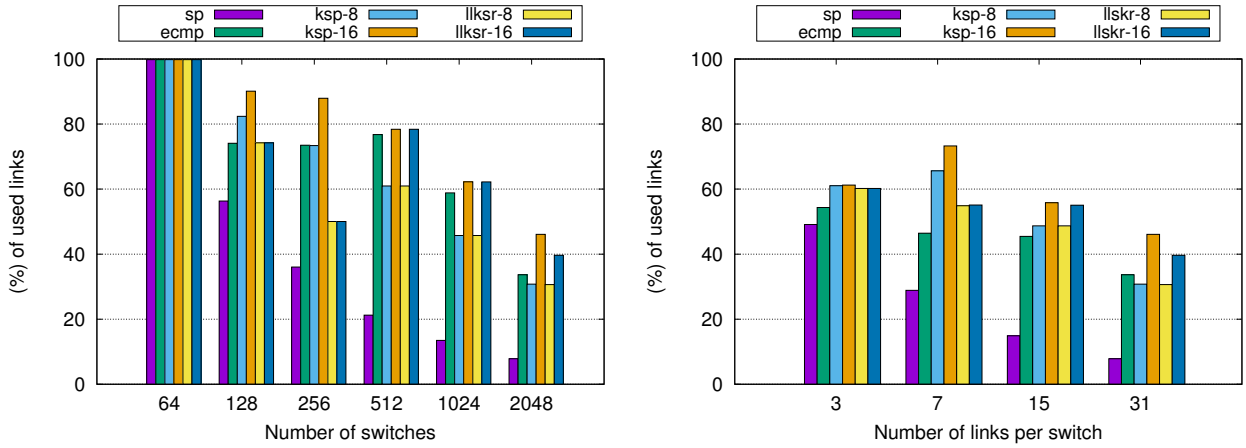
## 3.1 One application's Link utilization

The development of new routing policies for jellyfish was motivated due to the incapacity of classic policies to take advantadge of the high number of paths connecting every pair of switches. As a results, these policies, such as ksp or llskr, provide a great diversity for chosing paths connecting every pair of communicative severs. However, these policies have been only evaluated in mono-application scenarios where different applications do not compete for the use of the network resources. It is in those scenarios, where these policies could degrade the performance of the applications due to the high number of links used and hence, the increase share of the network links. In addition, the lack of causality in the communications used in those evaluations, does not represent the execution of real applications and , as we will see, it is an important performance facto in multi-applications scenarios.

Let us start analyzing the number of links that an application composed of 64 tasks uses when performing all-to-all communications in networks with different number and types of switches. In particular we have used those summarized in Table 1. We have used sequential and random allocations, although the results are almost identical, we only have represented the former. The results will give us a hint of how much interference one application could create in the network and therefore affect the performance of other applications. Results are depicted in Figure 2.

As we can see in Figure 2a the number of used links is determined by the routing policy. The 65-task application in the smaller network uses all the links regardless of the used routing policy. However, as the size of the network increases, the number of used links also increases except for SP (XXX data). As a result, SP will generate less interference with other applications but, at the same time, provide less path diversity that could result in higher contention in the links and therefore less performance.

Regarding the other policies, it is clear that




(a) Networks with differen size.

(b) Networks with different connectivity.

Figure 2: Percent of used links by a 64-task all-to-all application using a sequential allocation

Table 1: Network topologies used to measure the number of used links.

| Network | # Links |
|---|---|
| RRG(64,32,31) | 2112 |
| RRG(128,32,31) | 4224 |
| RRG(256,32,31) | 8448 |
| RRG(512,32,31) | 16896 |
| RRG(1024,32,31) | 33792 |
| RRG(2048,4,3) | 10240 |
| RRG(2048,8,7) | 18432 |
| RRG(2048,16,15) | 34816 |
| RRG(2048,32,31) | 67584 |

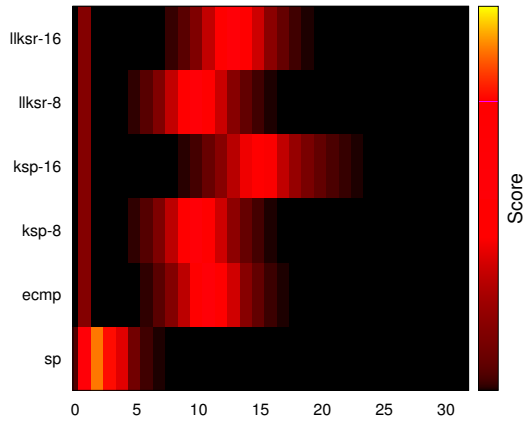



Figure 3: Number of shared links used by multiple applications when running an all-to-all communication pattern using different routing policies.

## 3.2 Multiple application's link utilization

After measuring the way the routing policies affect the to the number of used links by one application let us see the influence of the routing policies in multi-application scenarios. To do so, we run multiple applications concurrently and measure the number of links shared by how mnay applications. These results allow us to quantify how much interference will generate each routing policy. The results are depicted in Figure 3 where the execution of 32 concurrent 64-task applications in a RRG(2048,32,31) network is represented.

As espected, SP creates the lower contention among different applications being six the maximum number of application sharing links. However, the number of shared links is much higher than using other routing policies and could results in degraded applications performance. In contrast, if we focus on the other routing policies it is clear that different application shre much more links but the times those links are used are fewer.

Other interesting result is that as we increase the number of paths in the *ksp* and *llskr* policies the number of applications sharing communication links increases. In particular if we compare ksp-16 and llskr-16 we can see the effect of using shorter paths by the latter that result in less application sharing links due to the more *communications*.

# 4. LOCALITY AND CONTIGUITY IN JELLYFISH

As we have seen, the execution of concurrent application in jellyfish causes much interference between them. As a result, the perfomance of them can be affected. This effect has been widey studied for other topologies and the solution was the introduction of locality-aware allocation policies. The goal of these policies is to maintain the task composing the applications physically close in the network, thus reducing the presence of long paths, in order to avoid that contention.

In this section we adapt the idea of those policies to jellyfish proposing two types of allocation policies: contiguous that avoids any interference with other applications and pseudo-contiguous that relaxes the constraint of the contiguous version allowing some interference between different applications.

## 4.1 Contiguous allocation

## 4.2 Pseudo-ontiguous allocation

# 5. EXPERIMENTAL SET-UP

## 5.1 Simulation environment

## 5.2 Communication patterns

## 5.3 Performance metrics

# 6. ANALYSIS OF THE RESULTS

# 7. CONCLUSIOSN ANS FUTURE WORK

# 8. REFERENCES


[1] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, (Berkeley, CA, USA), pp. 225–238, USENIX Association, 2012.

[2] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[3] X. Yuan, S. Mahapatra, W. Nienaber, S. Pakin, and M. Lang, "A new routing scheme for jellyfish and its performance with hpc workloads," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, (New York, NY, USA), pp. 36:1–36:11, ACM, 2013.

[4] J. A. Pascual, J. Miguel-Alonso, and J. A. Lozano, "Optimization-based mapping framework for parallel applications," *Journal of Parallel and Distributed Computing*, vol. 71, no. 10, pp. 1377 – 1387, 2011.

[5] J. Navaridas, J. A. Pascual, and J. Miguel-Alonso, "Effects of job and task placement on parallel scientific applications performance," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pp. 55–61, Feb 2009.

[6] J. A. Pascual, J. Navaridas, and J. Miguel-Alonso, *Effects of Topology-Aware Allocation Policies on Scheduling Performance*, pp. 138–156. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.