Isolated Trees in Multi-Tenant Dragonfly+ Datacenters for In-Network Computing

Ori Rottenstreich, Gal Grimberg , Aviad Tal and Daniel Ben Hayoun

I. Introduction

A. Background

Our work is a continuation to a previous article " Isolated Trees in Multi-Tenant Fat Trees Datacenters for In-Network Computing", we will focus on the Dragonfly+ topology. Our goal is an algorithm that solves Problem 1 - Joint Mapping and EDH Trees Selection. That is, given a network topology and requested tenant demands (number of hosts per tenant), find a mapping of tenants to hosts and its corresponding Edge Disjoint Host Trees that maximizes the number of tenants with trees. Unlike the Fat Tree topology, it is not possible to guarantee mapping for all the tenants requests with DragonFly+. The radix determines the number of groups and the amount of hosts each group contains.

B. What is Dragonfly+

Dragonfly topology was introduced by Kim et al. aiming to decrease the cost and diameter of the network. The topology divides routers into groups connected by long links. Each group strives to implement high-radix virtual router, connected by a fully-connected topology. Dragonfly+ is an extension of the Dragonfly topology. It defines the topology of the intra-group routers and connects them as a bipartite graph. The main benefit of Dragonfly+ is the network scale. A network made from 32-port routers can support up to 105,000 hosts, as opposed to 26,000 with regular Dragonfly. In our work, we focused on Medium-size Topology of Dragonfly+, which means every spine router is connected to every other group by a single global link.

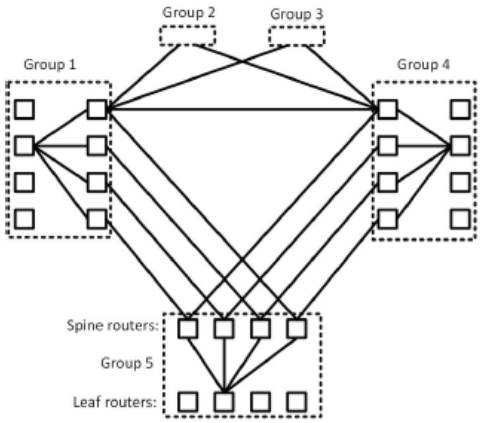


Fig. 1. Dragonfly+ medium

The number of hosts in each group is defined by the radix (notated). Every Leaf router connects to $\frac{r}{2}$. Which means the total available hosts for the topology is defined by: $N_{groups} = \frac{r^2}{4}$. In medium topology, the number of groups (notated) is also defined by the radix: $G = \frac{r}{2} + 1$ Which means the total available hosts for the topology is defined by: $N_{hosts} = G * N_{groups} = \frac{r^3}{8} + \frac{r^2}{4}$

II. Our Trace

The trace we use includes 16883 requests which indicated their number of required hosts. While 57.8% of the tenant requests were of a single host, 81.1% asked for up to 64 hosts and less than 3% of them were of more than 2048 hosts. Taking N_{hosts} into account, any radix of 25 or higher could satisfy more than 97% of the tenant requests. The largest request in the trace was for 9399 hosts, which would require a radix of 42 or higher to satisfy. However, searching for an allocation for a large tenant requires a lot of computation power due to the complex connectivity of long links between distinct groups (as opposed to the Fat Tree topology). In this article we suggest a 2-hop bound for each tenant to reduce run time. This means that each tenant can span on three groups at most, which limits the maximum host requested to: $tenant_limit = \frac{3r^2}{4}$.

Since the evaluated Dragonfly+ network has a limited number of hosts (determined by radix), we consider only the requests following such bound on their size. We try to schedule requests while keeping their arrival order such that a tenant is served when its required number of hosts is available. The total number of tenants active in parallel when a new tenant is scheduled can be quite large when tenants are small. As mentioned, most of the tenants are of a single host. Since a tenant with a single host never requires a tree allocation, the number of tenants with at least two hosts was also examined.

III. Dragonfly+ Tree Allocation Algorithm

Due to the complexity of the DragonFly topology we had the following assumptions:

(i) Tenants are never allocated trees that span over more than 3 groups.

(ii) The spines of each groups are indexed and are connected to spines of the same index of each of other group.

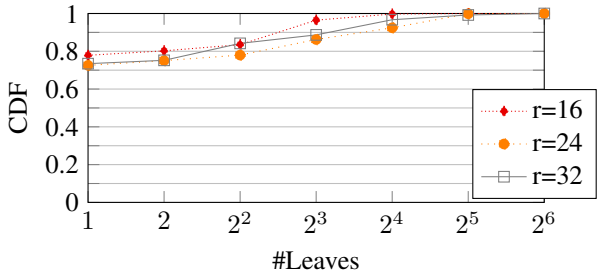
The algorithm attempts to find a type I tree for each tenant with as little long-links as possible. To optimize the placement, the algorithm distinguishes between small tenants of size [S] and bigger tenants, where S is equal to $\frac{radix^2}{4}$ which is half the amount of hosts in each group. Pseudocode of the algorithm

Data: Topology $G = (V, E)$, demands Φ with number of requested hosts N_i^{Φ} for tenant i
Result: Tree set F for tenants
$F = \emptyset$
for $i \in [1, t]$ **do**
  Placement = False
  while *!Placement* **do**
    Sort all groups by the number of available hosts
    Find a type I tree T in group g that can contain tenant i
    if T *not found* **then**
      Find a type I tree T in two groups g,g' that can contain tenant i
    end
    if T *not found* **then**
      Find a type I tree T in three groups g,g',g'' that can contain tenant i
    end
    if T *found* **then**
      Set Placement to True
    end
    if T *not found* **then**
      Wait until another tenant finishes
    end
  end
  if T *not found* **then**
    Return Failure
  end
  $F = F \cup \{T\}$
end
Return F

Algorithm 1: Dynamic Tree Allocation for Multiple Tenants by Long Link Minimization

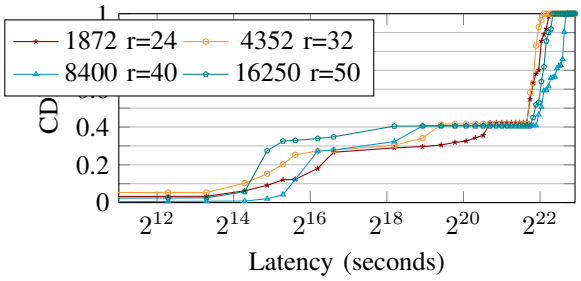
can be found in Algorithm 1. First sort all groups by the number of available hosts. Then check whether a one-leaf can be allocated. then check if a type I tree can be allocated among the sorted groups, using only one group. If such a tree cannot be built, check for a type I tree containing two groups. Last, check for a type I tree containing three groups. If a tree cannot be found, wait until another tenant finishes and try again.

IV. Experimental Evaluation

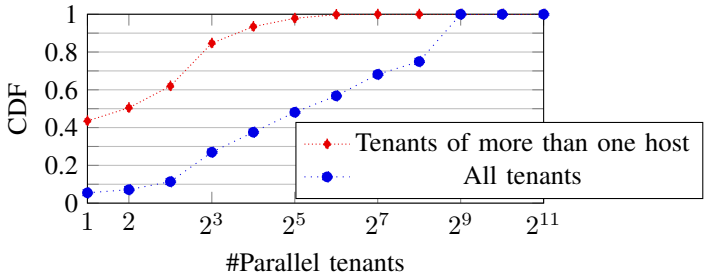


(a) Leaves per tenant

First, in Fig. 2a, we compare the number of leaves with at least one host of a tenant for varying radix size. We observe



(b) Latency distribution (over all trace)



(c) Latency over time

Fig. 2. Latency of tenants for various computation power (# hosts) and unrestricted number of spines

that with 576 hosts (radix = 16) the bound tenants require less leaves but cover only 84% of the trace. With 4352 hosts (radix = 32) we reach 93% of the trace, but such tenants require more leaves. Furthermore, the experiment with a radix of 32 shows better results than the corresponding radix of 24 in terms of strain on the system despite the larger tenants served. This is due to the increased size of the topology.

Next, we study the time a request with its required number of hosts has to wait before being served. In figure 2b We examine various radix numbers in the topology implying different total numbers of available hosts. We examine radix $r \in [24, 32, 40, 50]$ such that the number of hosts satisfies $h \in [1872, 4352, 8400, 16250]$. For example, in a topology with radix of 24, 15% of the tenants waited less than 2^{16} seconds, which means 85% waited more than that. With a radix of 50, the number of tenants that waited more than 2^{16} seconds is reduced to 67%.

We schedule requests while keeping their arrival order such that a tenant is served when its required number of hosts is available. Figure 2c shows the CDF of the total number of tenants active in parallel when a new tenant is scheduled (lower curve) on a test with radix = 24. This number can be quite large when tenants are small. As mentioned, most of the tenants are of a single host. Since a tenant with a single host never requires a tree allocation, the number of tenants with at least two hosts is also presented (upper curve). For example, 30% of the tenants ran in parallel with 8 or less other tenants. If we focus on tenants with more than one host, the number reduces significantly - the majority of the tenants (85%) run in parallel with 8 or less other tenants. Average parallel tenant numbers are 379 for all tenants and 12 when

considering tenants with more than 1 host.

V. Conclusions and Future Work

We developed an algorithm for isolation of tenants in medium-sized Dragonfly+ topology through the allocation of EDH trees. Analytical results and experiments showed that the availability of such trees correlates strongly with the radix size and the number of hops allowed. With all the assumptions and limitations, the algorithm still managed to serve a considerable percentage of the tenants in the trace, providing comparable results to that of the Fat Tree. Comparing to the Fat Tree with host size of 2048, a Dragonfly+ topology with radix of 24 has the closest number of hosts. However, the suggested algorithm only accepts tenants with size of 432 for that radix, because it allows up to two hops (or three groups). A Dragonfly+ topology with radix of 52 has 16,250 hosts and can serve tenants requests of size 2028 (which is closest to the original Fat Tree example). Such a topology would out-perform the original Fat Tree because of the substantial difference in number of available hosts, but the cost would be significantly more expensive. An improved algorithm that allows more hops could bring comparable results to those of the Fat Tree but will require increased computational power. We predict it is possible to develop a learning algorithm that optimizes the distribution of a tenant's hosts over multiple groups while maintaining a feasible runtime.