Jigsaw: A High-Utilization, Interference-Free Job Scheduler for Fat-Tree Clusters

Zach Hegemann and Patrick Berry

Jigsaw: An Overview

- Jobs on HPC clusters can suffer significant performance degradation due to inter-job network interference.
- Different approaches to mitigate this interference typically focus on reactive routing schemes.
- Implementing scheduling policies that proactively enforce network isolation for every job can completely eliminate this interference. Existing schedulers typically have lower system utilization, however.
- We present Jigsaw, a job-isolating scheduling approach for three-level fat-trees that achieves 95-96% utilization.

Jigsaw Introduction

- Schedulers on most HPC clusters allocate dedicated nodes to each job, but do not take network resources into account. This leads to multi-node jobs competing for resources on the network.
  - This can lead to substantial job performance degradation.
- Having a job scheduler enforce complete network isolation is, conceptually, an elegant way to alleviate inter-job network interference. This comes with challenges, however:
  - Ensuring performance isn't degraded by guaranteeing each job has access to the same underlying bandwidth in its isolation.
    - This requirement leads to constraints on job-to-node assignment and thus system utilization.
- Jigsaw is a scheduling approach that removes the system utilization barrier.
  - Simultaneously achieves job-level network isolation, full bandwidth of job allocations, and high system utilization.

Jigsaw Background: Fat-Tree Networks

- A fat-tree is a tree whose links have higher bandwidth at each level going up from leave to root.
- Because the link bandwidth increases, the links near the root of the tree do not create a bottleneck when nodes communicate across the tree

Jigsaw Background: Inter-Job Network Interference

- Inter-job network interference has been identified as a culprit behind job performance variability on HPC systems.
  - On torus- and dragonfly- topologies, it can cause applications to slow down by 100-150%.
  - Fat-tree networks with multi-job workloads, under static routing, can slow down by as much as 120% in controlled experiments and up to 66% in production applications.
- Exploring techniques to mitigate inter-job network interference is paramount

Jigsaw Background: Existing Mitigation Approaches

- Routing-Based Approaches
  - Route packets in a way that reduces congestion *after* job placements are already fixed. An advantage of this is that there aren't any scheduling constraints, leaving node utilization unaffected.
  - These routing techniques cannot guarantee that the worst-case performance degradation for jobs is small.
- Scheduling Approaches
  - Inter-job network interference can be completely eliminated by using a scheduling policy that guarantees isolated network partitions for each job.
  - Job-isolating scheduling requires new node placement constraints, and these can lead to system fragmentation and lower node utilization.
    - Previous job-isolating approaches, node utilization dropped by ~10%.

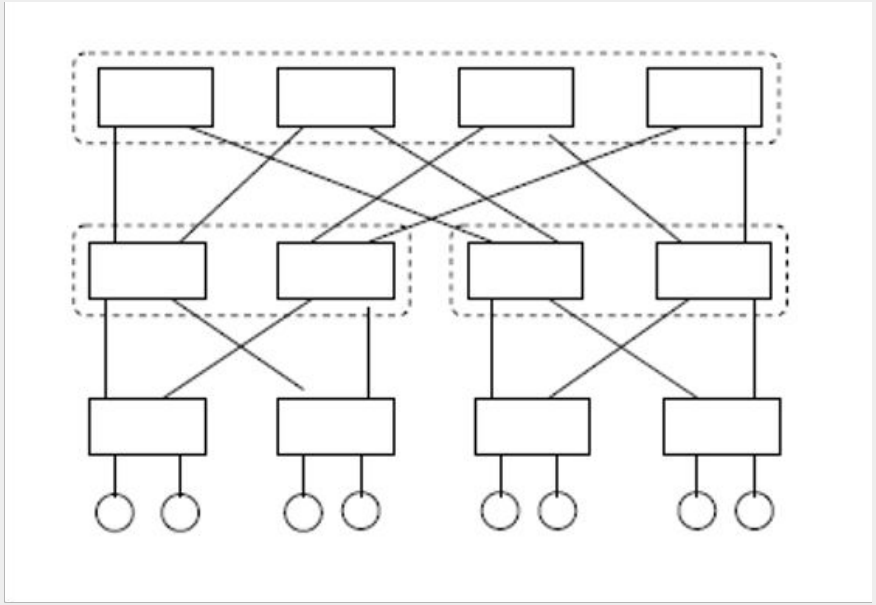
Jigsaw Theory: Motivation

The goals of the Job Scheduler

a) The scheduler must allocate isolated partitions that are free from inter-job interference
b) The scheduler must allow access to full interconnect bandwidth inside each partition
c) The scheduler strives to provide high utilization- at least 95%- to keep high throughput

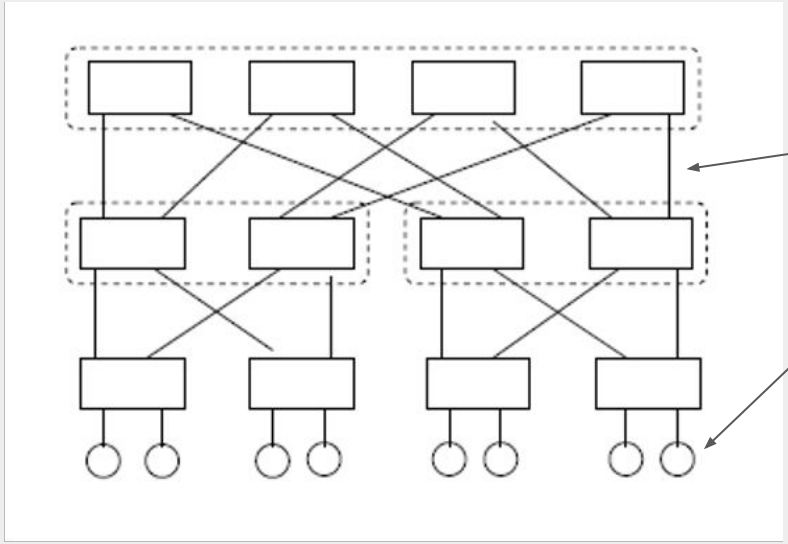
Satisfying any two of the three goals is straightforward.

The challenge is achieving all three at once!



Jigsaw Theory: Motivation

Isolation: ensures that jobs do not interfere with each other on the network. This requires that each node and each link be (exclusively) assigned to at most one job. Most existing job schedulers ensure only node isolation.



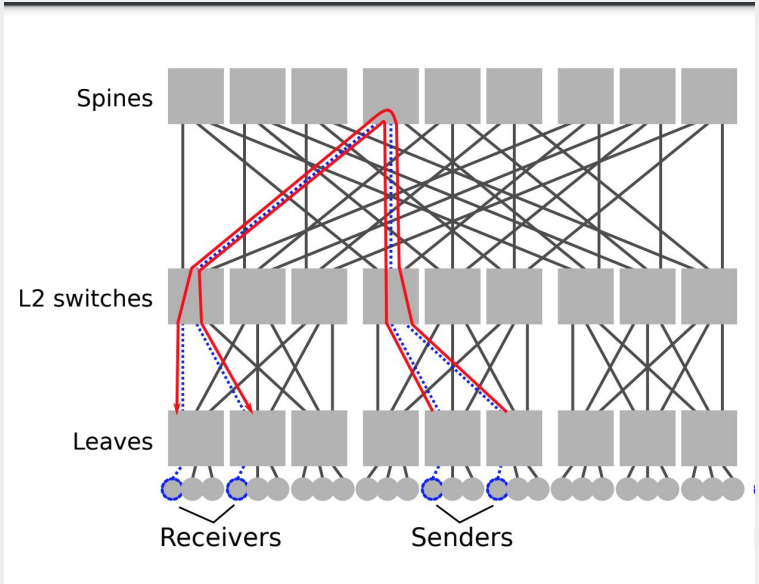
Jigsaw Theory: Motivation

Full interconnect bandwidth: ensures that an allocated partition has the bandwidth properties of the fat-tree itself

rearrangeable non-blocking: any permutation of traffic among the nodes of a job can be routed such that only one flow travels over any of the job's links

Don't share paths/links

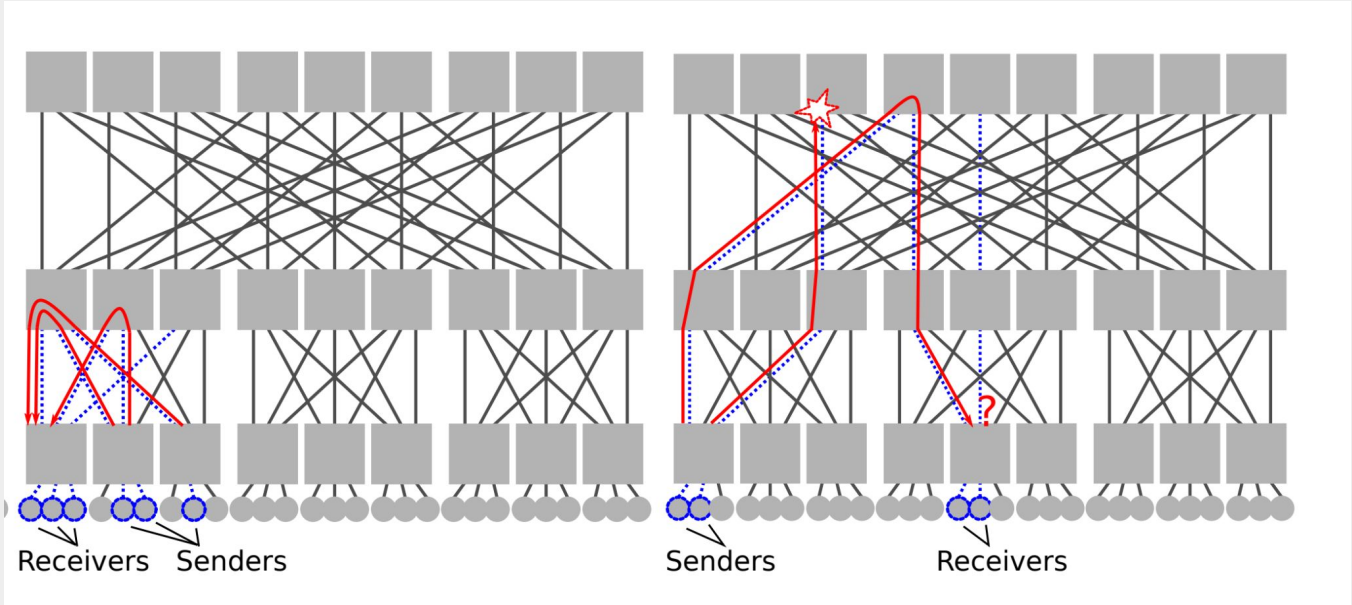
1. Uplinks != Downlinks is no good



Jigsaw Theory: Motivation

2. Cannot allow full generality in node-to-job assignment. Two flows are forced to share the left-most link in the example

3. Balanced uplinks and downlinks have been selected independently at each switch. Although a sufficient amount of links, they are wasted because they cannot reach the edge. Poorly chosen.



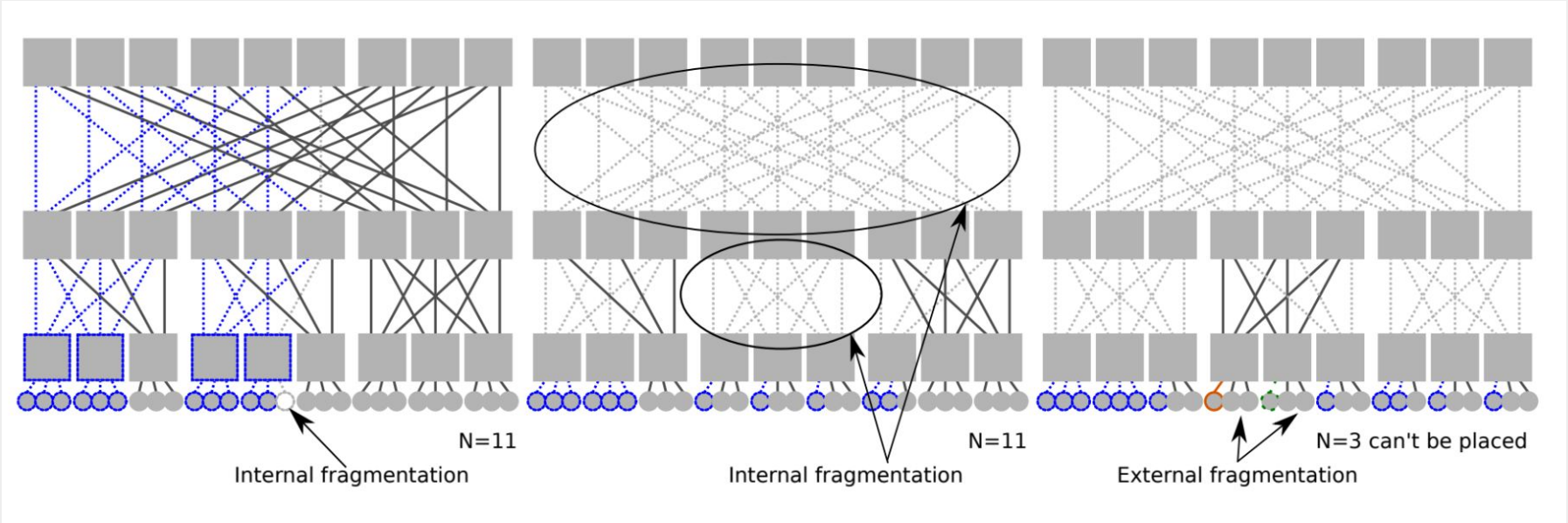
Jigsaw Theory: Motivation

High Utilization: This generally helps in reducing fragmentation of nodes and links. The challenge is to provide high utilization while maintaining the isolation and full bandwidth constraints.

1. Internal fragmentation of nodes or links occurs when a given scheme requires that a leaf allocate all nodes or links to the same job, but the job does not use all nodes or links
2. External fragmentation of nodes or links occurs when there are enough nodes or links for a job, but making the allocation violates node and link conditions

JigSaw Theory: Motivation

Examples of Fragmentation, reducing the system's utilization



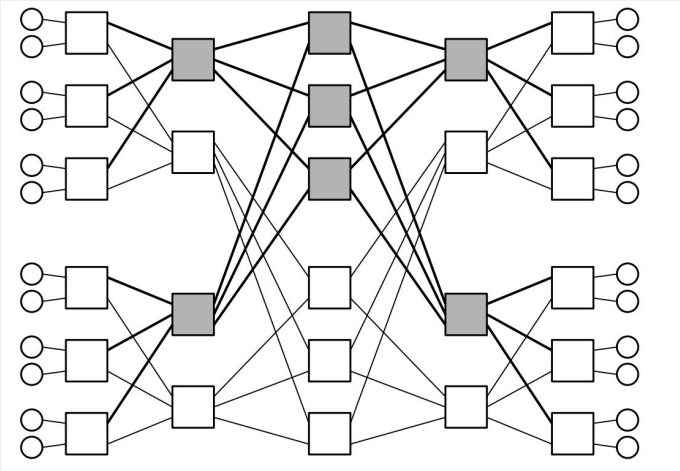
LAAS approach

TA approach: links are reserved for the first job that can physically reach them

TA approach: requires that a job must be assigned to single leaf if it can fit

Jigsaw Theory: Formal Conditions, Isolation

- We note that a three-level fat tree is composed of a set of independent two-level fat-trees connected together at the third level by spine switches. For brevity, we refer to these two-level subtrees as simply trees for the remainder of this section
- Isolation
  - nodes i and j are assigned to two different jobs, then i != j
  - links k and l are assigned to two different jobs, then k != l

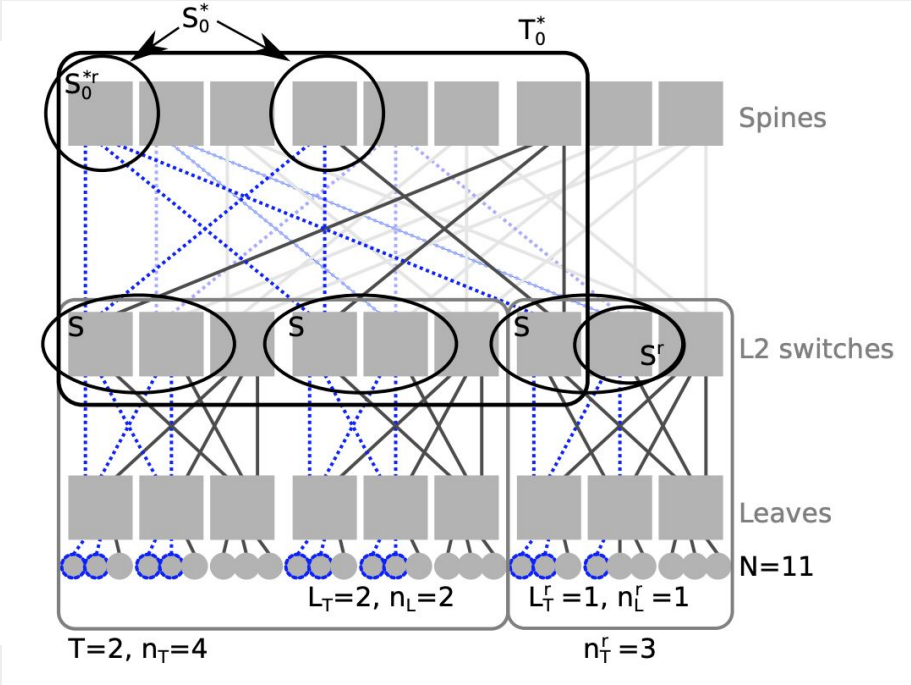


Jigsaw Theory: Formal Conditions, Full Bandwidth

(1) The N nodes of a job must be evenly distributed across T trees with n_T nodes each, plus an optional remainder tree with $n_T^r < n_T$ nodes.

(2) Within each of the T trees, the nodes must be distributed across L_T leaves with n_L nodes each. In the remainder tree, they must be distributed across L_T^r leaves that also have n_L nodes each, plus an optional remainder leaf with $n_L^r < n_L$ nodes.

(3) The nodes must be allocated across a set of identical trees and one remainder tree; and all leaves in the allocation must have the same number of nodes except a single remainder leaf in the remainder tree. Note that $N = T \cdot n_T + n_T^r = T(L_T \cdot n_L) + (L_T^r \cdot n_L + n_L^r)$ by these conditions.

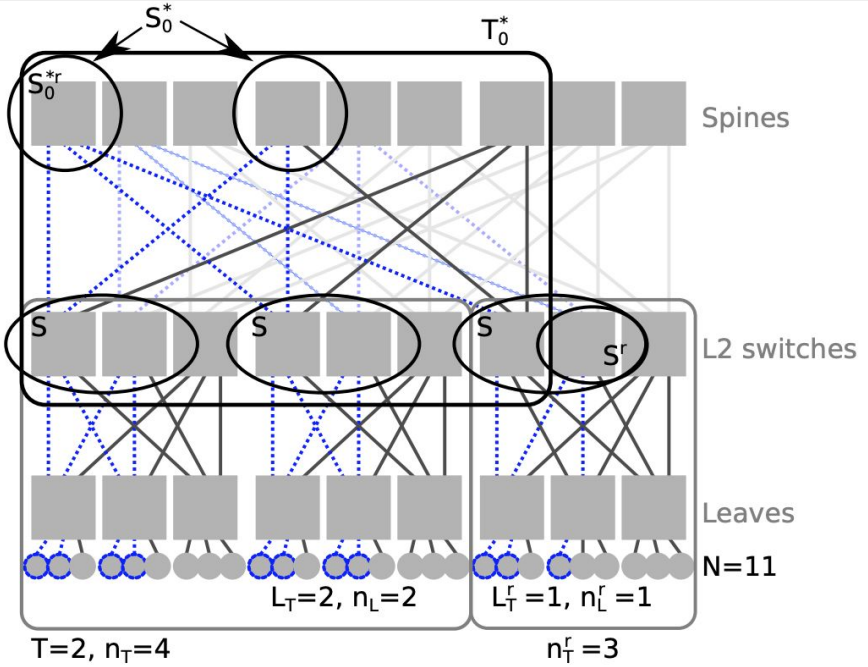


Jigsaw Theory: Formal Conditions, Full Bandwidth

(4) Within each allocated *two-level* tree, the L_T (or L_T^r) leaves must connect to a common set of L2 switches S, while the remainder leaf must connect to a set of L2 switches $S^r \subset S$.

(5) Each of the allocated trees must use a consistent set of L2 switches S; which means that in every tree, the set S must contain L2 switches at the same set of indices within the tree. Formally, each tree S must contain the L2 switches at i_1, i_2, ..., i_{n_L} for some set of i_k that is common across all trees in the allocation.

(6) At the top level, the i^{th} L2 switch of each tree is part of a full-bipartite graph with a subset of the spines; we denote as T_i^* this set of L2 switches, spines, and links. The partition T_i^* has the same structure as a two-level fat-tree, and so it has conditions similar to (2) and (4). Namely, if the i^{th} L2 switch of an allocated tree is in S, then it must be allocated a balanced number of uplinks to the spines; each such switch must connect to the same set of spines S_i^*, except if it is in the remainder tree. In this case it must connect to a subset of the spines $S_i^{*r} \subseteq S_i^*$.



Jigsaw Theory: Formal Conditions, Full Bandwidth

At the node level, the number of nodes assigned to a job must be exactly the number of nodes that the job requested. Denoting the number of requested nodes by Nr and the number of assigned nodes by N, this means that N = Nr . At the link level, every leaf and L2 switch must be allocated the same number of uplinks as downlinks

Jigsaw Theory: Full Bandwidth Proof Sketch

The previous conditions are both necessary and sufficient for a job's allocation to be rearrangeable non-blocking.

- A network is *rearrangeable non-blocking* if, for any permutation of traffic among its nodes, there exists a routing that maps at most one flow of traffic to every link.

Jigsaw Theory: Full Bandwidth Proof Sketch

- To prove the necessity of the conditions it is shown that if each individual condition does not hold, then the allocation cannot support a particular traffic permutation among its nodes without contention
- Specifically, two subsets of nodes of size n are picked. It is shown that if one subset sends n flows to the other, they will be confined to fewer than n links.
    - The extended version of the paper proves that, if all conditions are not met, flow conflicts always occur!

Jigsaw Theory: Full Bandwidth Proof Sketch

- To prove the sufficiency of the conditions, an arbitrary partition that satisfies them is taken and shown that an arbitrary permutation can be routed with at most one flow per link.
    - This is done by viewing the three-level-fat-tree as an equivalent Clos network.

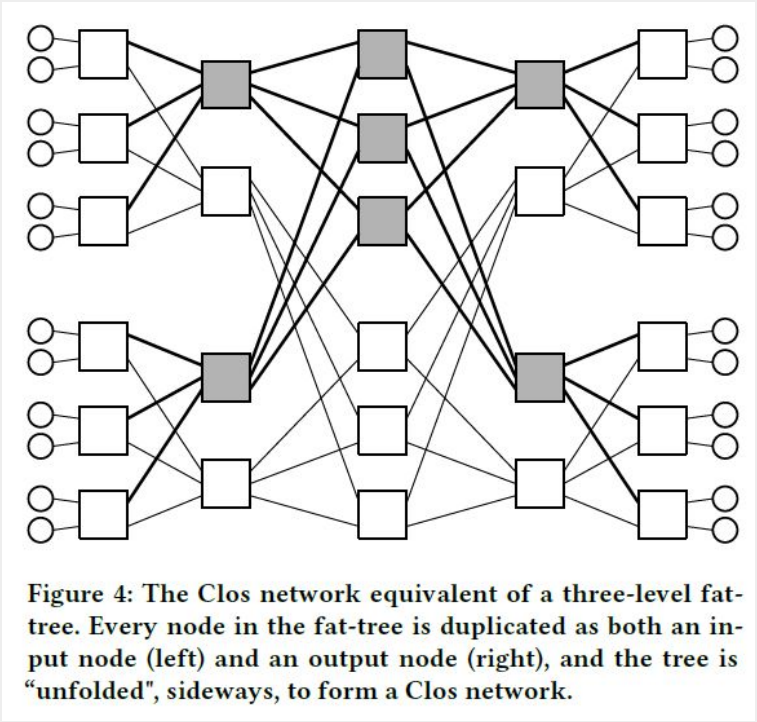


Figure 4: The Clos network equivalent of a three-level fat-tree. Every node in the fat-tree is duplicated as both an input node (left) and an output node (right), and the tree is "unfolded", sideways, to form a Clos network.

Jigsaw Theory: Full Bandwidth Proof Sketch

- The general method of the proof is inspired by the existing result that two-level fat-trees are rearrangeable non-blocking.
- Suppose P is an arbitrary permutation of input nodes to output nodes, such that the the flow from input to output nodes forms a bijection.
    - According to Hall's Marriage Theorem, there exists a subset of flows in P such that the subset contains exactly one flow coming from each leaf and one flow going to each leaf.

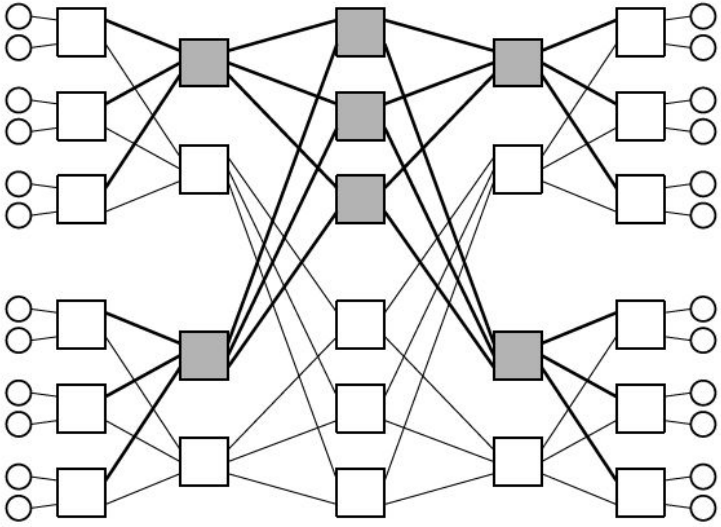


Figure 4: The Clos network equivalent of a three-level fat-tree. Every node in the fat-tree is duplicated as both an input node (left) and an output node (right), and the tree is "unfolded", sideways, to form a Clos network.

Jigsaw Theory: Full Bandwidth Proof Sketch

- All flows in the subset may be routed across the same part of the inner-network (ex. the grey network in Fig 4.), with each link at the first and last stage carrying exactly one flow.
    - The inner-network is a smaller Clos network, equivalent to a two-level fat-tree.
    - As a two-level fat-tree is rearrangeable non-blocking, this subset of flows is satisfactorily routed across the three-level fat-tree.
- This subset is then removed from the network (leaving behind white switches), and the process is iterated until every flow is routed.

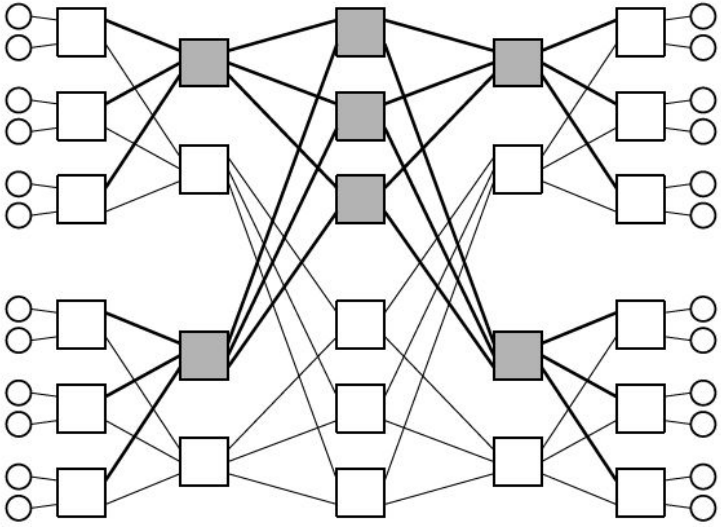


Figure 4: The Clos network equivalent of a three-level fat-tree. Every node in the fat-tree is duplicated as both an input node (left) and an output node (right), and the tree is "unfolded", sideways, to form a Clos network.

Jigsaw Implementation

Still a few more issues….

1. The conditions we found allow a wide variety of legal allocations—the number of possible allocations is exponential in the size of the tree, making an exhaustive search for placements infeasible for large systems
2. No prediction on free nodes on a given leaf, fragmentation - For a large job, there may be limited number of options

The solution …

1. Jigsaw requires job allocations that span three levels to use all nodes per leaf except for the remainder leaf

Jigsaw Evaluation Setup

- To evaluate different scheduling approaches, job queues on various fat-trees were simulated.
- Jigsaw is compared to several other job-isolating scheduling approaches and to *Baseline*: a traditional, unconstrained scheduler. Evaluation is measured in terms of average system utilization, given by

$$U = \frac{\text{time spent by jobs on all nodes}}{\text{potential time on all nodes}} = \frac{\sum_{j=1}^{\text{jobs}} N_j \cdot t_j}{N_{\text{system}} \cdot t_{\text{total}}}.$$

- Nj and N{system} are the number of nodes in job j and the total number of nodes in the system, respectively; tj and t{total} are the runtime of job j and total time spent running jobs in the trace, respectively.

Jigsaw Evaluation Setup: Clusters and Traces

- Evaluation was done on a range of full fat-trees with switch radix varied to attain different node counts.
  - Experiments were ran with four different cluster sizes: 1024, 1458, 2662, and 5488.
- Job queue traces from several LLNL clusters were used, including Thunder and Atlas as well as Cab. Synthetic traces were also used; generated in the same way as the ones in the original LaaS paper.

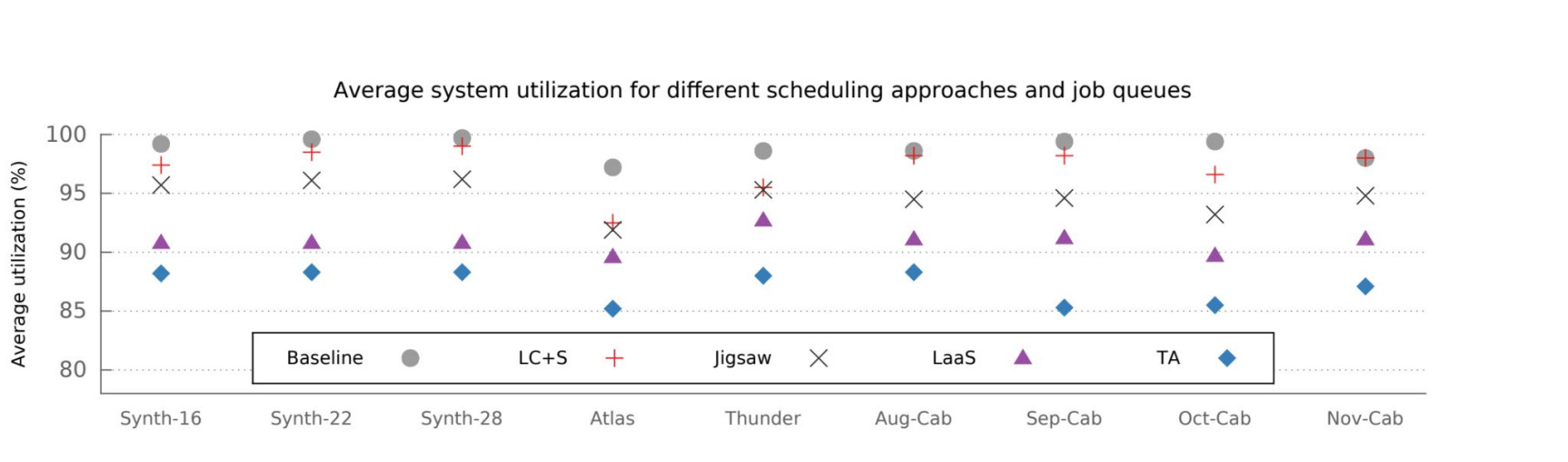
Jigsaw Evaluation Setup: Schemes for Comparison

- Jigsaw was compared to three approaches:
  - Links as a Service (Laas): a scheduler that allocates dedicated network links (and nodes) to each job. LaaS, unlike Jigsaw, does not develop explicit placement conditions for three-level fat-trees.
  - Topology-Aware Scheduling (TA): links are not explicitly allocated to jobs. Instead, TA follows a set of node allocation rules to avoid all placements in which two jobs could contend for links under arbitrary routing.
  - Least-Constrained Scheduling with Link-Sharing (LC+S): allows jobs to share network links as much as possible while maintaining interference levels that are expected to be negligible.

Jigsaw Evaluation Setup: Experiment Parameters

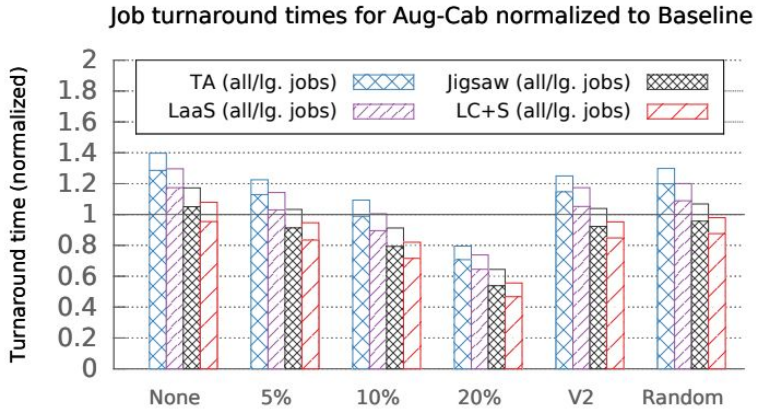
- Job Performance Scenarios: when job turnaround times and makespan are evaluated, the fact that some jobs will likely perform better when ran in isolation is taken into account.
  - Four scenarios from the TA scheduling paper (5%, 10%, 20%, and “V2” scenarios) are used.
- Link-Sharing Parameters: For the LC+S experiments, average bandwidth needs for each job must be determined.
- General Parameters:
  - Three synthetic traces have mean job sizes of 16, 22, and 28, and they’re simulated on 1024-, 2662-, and 5488-node clusters, respectively.

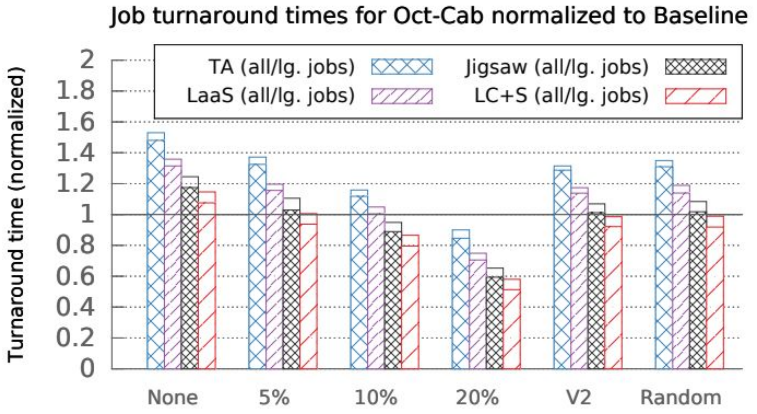
Jigsaw Results: Average System Utilization



Jigsaw Results: Average Job Turnaround Time

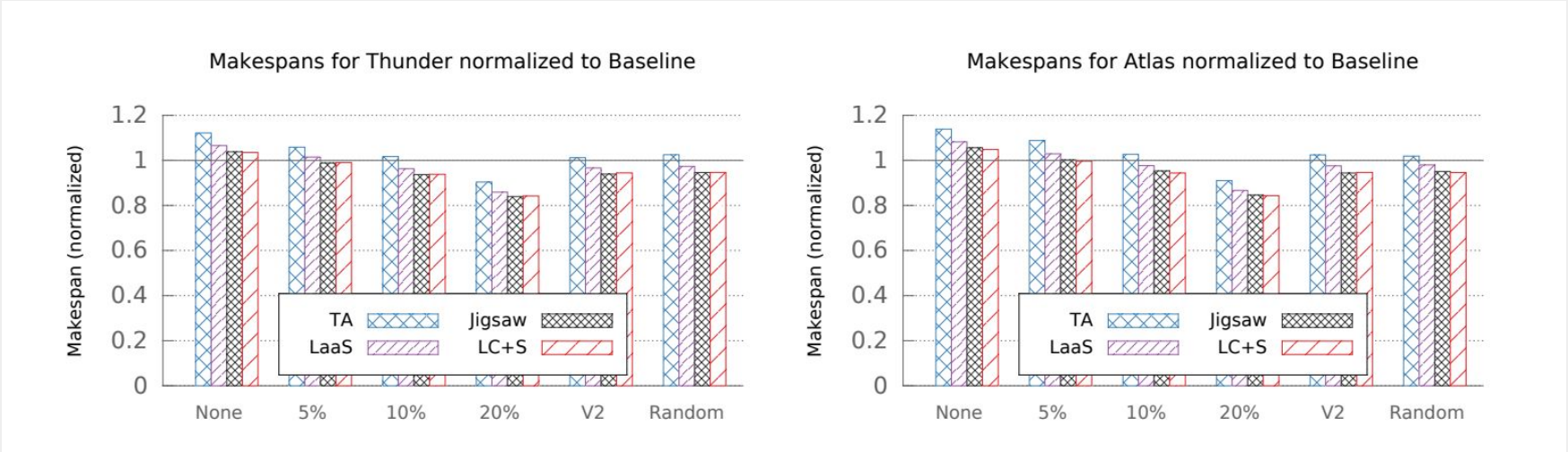
Job turnaround time is the length of time between when a given job arrives in the queue and when it finishes running





Jigsaw Results: Average Job Makespan

Makespan is the length of time between when the first job arrives in the queue and when the last job running on the system completes



Our Jigsaw Conclusion

Since each job is guaranteed network isolation, application developers can focus their efforts on optimizing intra-job network performance without worrying about network traffic outside their control, and performance variability due to inter-job network interference is eliminated

Questions ? ? ?