A Study into the Effects of Topology Switching for Scalable Client-Server Distributed Applications

Alexander Summers

ABSTRACT

1 INTRODUCTION

1.1 Motivation

The exponential growth of client-server distributed applications has led to a need for them to be scalable systems able to service many users. Many applications currently rely on static configurations or topologies, which could limit their ability to change scale and adapt to changes in traffic. This inflexibility potential could cause increased latency and even service outages. This is because static topology configurations can perform well overall depending on the situation; however; for the internet, there is not a one size fits all solution, and a simple switch of topology to compensate for weaknesses of the previous could lead to a tremendous performance and robustness increase. For example, popular client-server distributed applications like Zoom struggled during the early COVID-19 pandemic because their current infrastructure limited their scalability, fault-tolerance, and efficiency during the surge of the increased nodes and geographical distribution of nodes. Ultimately, this problem that plagued the video calling platforms was fixed using edge computing and CDNs; however, changing topologies could have also increased the distributed system's scalability.

This leads to the question: How does switching the topology in a distributed client-server setting affect the performance, and how it impacts the user's experience? It is imperative to address this problem to ensure the continued growth of client-server distributed applications to maintain user satisfaction.

Thus the novelty of our study comes from the approach to managing the topologies in a distributed setting, as other scalability solutions that have appeared, like CDNs and edge computing, do not relate to changing topology.

Suppose this study can conclude that switching a topology could provide performance boosts for an application. In that case, it opens up the way for developing solutions to dynamically manage the topology's structure to leverage those performance boosts. Therefore, increasing the performance and scalability of future applications. In addition, the implications of this study could be the increased use of client-server technology in distributed applications, as businesses may be interested in its increased performance and reliability. It could also contribute to the understanding and previously unknown insights into client-server networks. Lastly, It could be of value to researchers working on related topics leading to even more innovation in the future. In addition, if this study finds that changing the topology can give an increased performance to distributed applications, future work could be done to leverage the advantages.

1.2 Methods of Changing Topologies

Currently, as this study defines it [**?**], topology is reconfigurable using an Optical Transport Network (OTN) and routing protocols. The study says that the two primary methods of changing a server topology are Software-Defined Networking (SDN) and optical switches, which can change these fiber connections. SDNs, as defined by [**?**], are part of the network stack that decides the routing for packets for each router which, therefore, can change the network's topology by simply changing the routes for packets. OTN is a protocol that allows optical networks a reliable and efficient way of transporting data across a network. Furthermore, optical switches can change the topology by directly switching one optical signal to another. These are widely used in data centers, and more helpful information can be found at [**?**]. Thus, this study will explore the switching of topologies under these methods.

2 RELATED WORK

There has been exploration already into how optical switches affect data centers preformance done by [**?**]. They outline metrics for a datacenters preformance, such as scalability, flexibility, and performance, and introduce currently used topologies, fat-trees. The authors claim that optical switches will be the solution for scalability, listing out different types of switches and their speeds. This study claims that optical switches will increase scalability; however, it must demonstrate how effective enough for readers to be convinced. Therefore, our study aims to do more than the one previously described by actually conducting experiments to support the claims made by this paper.

In addition, an existing solution already explores dynamically changing topology [**?**] for distribution systems. The solution proposed by the authors is a step in the right direction for dynamic reconfiguration for a distributed application. Although, it only considers changing topology upon nodes joining and leaving the network. Also, the authors need to

evaluate their solution to a benchmark making the data they attained non-comparable or rightfully interpretable, leading their claims to be mere speculation. Therefore, before diving into a potential solution, it's essential to investigate which topology changes can enhance the system, and this is what our study aims to do, which is to explore what these beneficial changes are before actually making a dynamic topology switcher.

Existing solutions that explore testing different topologies, such as [**?**], explore working with a random-graph topology instead of a fat-tree topology to support more data center server nodes. They claim that using a random graph instead of a traditionally used fat-tree topology increases capacity by 25%. The authors show that their solution outperforms others in scalability and performance. Also, showing that their random graphs topology scheme could outperform fat trees. However, it only compares one topology to another in a data center setting. It needs to tackle the problem that this study is trying to solve, which is to see how switching a topology could affect distributed applications.

3 SYSTEM DESIGN

To test if changing the topology affects the performance of a distributed system, we will test multiple topologies with different client-server setups and then change the topology and compare the performance metrics between them. To accomplish this goal, we will simulate a distributed system on top of an actual topology from The Internet Topology Zoo called AGIS. For the distributed system that is being simulated, we are using the benchmark TPC-H made by the Transaction Process Preformance Council (TPC).

We use the AGIS topology for the simulation to reflect a real-world client-server system. Therefore, for transmitting data across the network, we only consider propagation delay calculated using the haversine distance between the coordinates of each edge of the graph.

The TPC-H benchmark comes with a database generator with business-related queries and data modifications that simulate distributed systems involving data warehousing, business intelligence, decision support systems, and online analytical processing. Therefore, the reason for including TPC-H is to simulate these groups of distributed systems in a distributed environment closely related to those in the real world.

The relevant metrics from TPC-H that will be used are throughput tests and queries per hour. The throughput test measure how fast multiple steams of queries can be executed. Lastly, the quries per hour convey how many queries can be satisfied over the course of an hour. These measurements are helpful in our study because we can compare the times

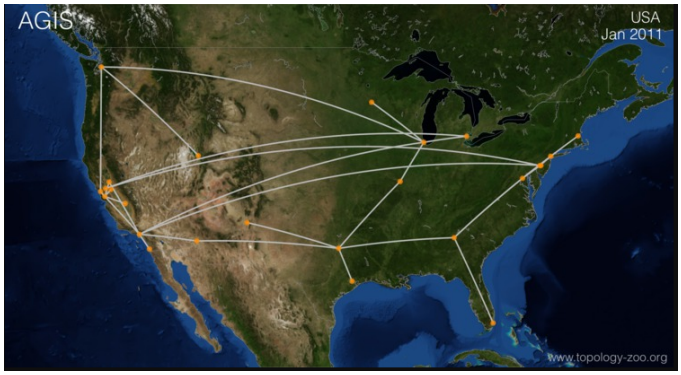

Figure 1: This is a picture containing the original AGIS topology.

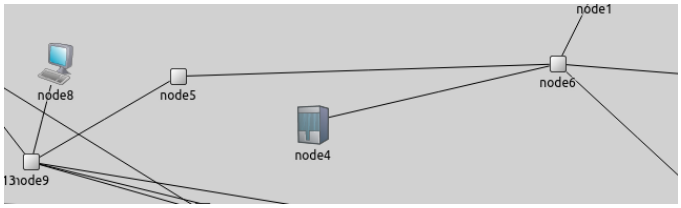

Figure 2: This is the one-server multi-client original topology where the clients are denoted as computers, and the server is shown as a server. Client node 8 is located at the top left and the server node 4 is shown to be in the middle.

and queries per hour with our original topology to another to conclude if there are any performance benefits.

Our proceeding simulations occur in OMNeT++ using an SQLite version of the TPC-H database converted by [**?**], and queries are manually tweaked to fit with SQLite. The server nodes that host the database sessions are also presumed to be single-threaded in these scenarios. All the source code for the simulations performed in this study can be found at [**?**].

The simulations performed in this study consist of a one-server, one-client scenario, a one-server multi-client scenario, and a multi-server multi-client scenario. Each will have an original and changed topology, with performance metrics for each that we will compare and contrast.

3.1 Simulations

3.2 One Server One Client Scenario

For the one server, one client scenario, we randomly assigned the server and client node to a random node in the system, creating our original topology, as shown in fig 2. We then extract performance metrics from the original topology for the throughput and queries per hour tests. We first conducted 10 throughput tests, which yielded an average time of 3.15s

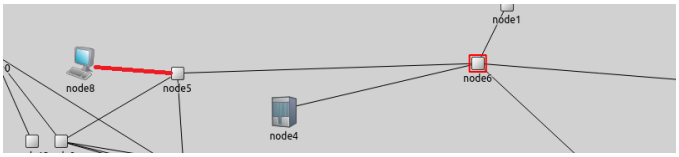

Figure 3: This is the one-server multi-client new topology where the clients are denoted as computers, and the server is shown as a server. Client node 8 is located at the top left, and the server node 4 is shown to be in the middle. Notably, the topology changed by removing the client's only connection and replacing it with a connection to node 5, reflected in red above.

Server Node	Old Topology QPH	New Topology QPH
Server 4	247.2	249

Table 1: Queries per hour at scale factor 0.01 for Server node 4 under old and new topologies in the one server one client scenario.

Topology	Average Time (Seconds)
Old	3.15
New	3.07

Table 2: Average time to for the server to service the client under old and new topologies in the one server one client scenario.

for the server to process all of the client's queries, as seen in Table 2. Then we conducted a simulation to record the queries per hour, which, with a scale factor of 0.01 on the database size, yielded queries per hour of 247.2 on the original topology, as seen in Table 1.

With the baseline metrics now set by the original topology, we need to see if we can change the topology to yield better metrics than our baseline. Therefore, we can reduce the latency between the client and server nodes by making the path between them shorter. Thus, by changing the client's connection to node 5, as seen in Fig 3, we will call this our new topology, rerun our simulations, and extract performance metrics for our new topology.

With the new topology, we conduct the same simulations and extract the same metrics as we did with the original topology. The 10 throughput tests yielded an average completion time of 3.07s, as seen in Table 2. The completion times of each client node in the throughput test are shown in Table 5. Lastly, the queries per hour on the new topology were 249, as seen in Table 1.

Comparing and contrasting the old topology to the new one, we can see that the new one performed slightly better

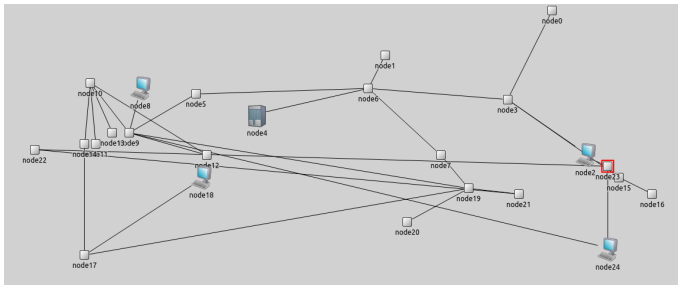

Figure 4: This is the one-server multi-client original topology where the clients are denoted as computers, and the server is shown as a server. Client node 8 is located at the top left, client node 18 is at the bottom right, client node 2 is in the middle right, and client node 24 is at the bottom right.

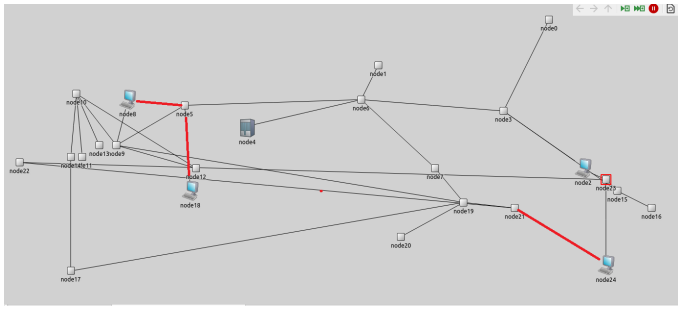

Figure 5: This is the one-server multi-client new topology where the clients are denoted as computers, and the server is shown as a server. Client node 8 is located at the top left, client node 18 is at the bottom right, client node 2 is in the middle right, and client node 24 is at the bottom right. The changes are that node 18 has a path to node 5, client node 8 connects to node 5, and lastly, client node 24 has a connection to node 21. All the changes can be seen in red.

based on the metrics. Specifically, with the new topology, the server can perform slightly more queries per hour and complete a set of queries from the client slightly faster. This provides sufficient evidence to claim that in a one client server distributed application, decreasing the latency between the client and server nodes by changing the topology increased the efficiency of the distributed system.

3.3 One Server Multi-Client Scenario

For the one server multi-client scenario, we consider a 4-client 1-server scenario; each is randomly assigned a geographic location, as shown in Fig 4, for which we will consider the original topology. For this topology, we simulated to extract metrics for the throughput test and queries per

Server Node	Old Topology QPH	New Topology QPH
Server 4	318.6	333

Table 3: Queries per hour based on all clients at a scale factor of 0.01 for Server node 4 under old and new topologies in the one server multi-client scenario.

Server Node	Old Topology (s)	New Topology (s)
Server 4	10.39	9.91

Table 4: Average time to service all clients for Server node 4 under old and new topologies in the one server multi-client scenario.

Client Node	Old Topology (s)	New Topology (s)
Node 2	7.74	9.22
Node 8	8.52	9.47
Node 18	10.39	9.91
Node 24	8.72	9.76

Table 5: Average time for each client node to complete its set of queries under old and new topologies in the one server multi-client scenario.

hour. We first conducted 10 throughput tests, which yielded an average time of 10.39s for the server to process all of the client's queries, as seen in Table 4. Also, notably, all the client's completion times for the throughput test are in Table 5. Then we conducted a simulation to record the queries per hour, which, with a scale factor of 0.01 on the database size, yielded queries per hour of 318.6 on the original topology.

Table 5 shows that node 2, node 8, and node 24 are relatively complete around the same time, while node 18 lags a noticeably large margin behind the other client nodes. Noticeably the client nodes with a closer travel distance to the server node have lower completion times. This is because, in this simulation, the server is single-threaded meaning it can only serve one client at a time, so the clients that are first to get in line will be serviced before others leading to them completing their set of queries faster than others. For example, client node 18 is the slowest to complete its queries because messages must travel inefficiently, going away from the server, as seen in Fig 4. Using this information, we can try to improve node 18's performance by repeating what we did with the last scenario making node 18's path shorter to the server node. Therefore, as seen in Fig 5, we can add a connection from node 18 to node 5 for the new topology, decreasing its travel distance.

In addition, the other client nodes could lower their latency with the server by adding connections to make the routes more efficient. For example, client node 8 could connect to node 5, like in the one-server scenario. Then, client node 24 could change its connection to node 21, potentially lowering its message travel times. These changes make up the new topology in Fig 5, which can be seen in red.

With the changes performed, we conduct the same simulations and extract the same metrics as we did with the original topology. The 10 throughput tests yielded an average completion time of 8.01s, as seen in Table 4. The completion times of each client node in the throughput test are shown in Table 5. Lastly, the queries per hour on the new topology were 333, as seen in Table 3.

Comparing the two topologies' performances, the queries per hour and throughput tests were noticeably better for the new topology. Specifically, the old topology averaged 318.6 queries per hour, while the new topology averaged 333, which is a slight improvement overall. Therefore, this simulation provides evidence that the server's queries per hour increase when the latency between the clients and server is decreased by making a shorter path for the client nodes to the server node. Continuing, for the average completion time for the server to complete all client node queries extracted by the throughput tests, the new topology performed around half a second faster than the old topology seen in Table 4. Therefore, for a multi-client setting, shorter paths between the client and server nodes lead to the server satisfying all client node queries faster. These both, in turn, benefit the performance of the distributed system overall hence showing that changing the topology can have a positive impact on a distributed system.

Another thing to notice is that in Table 5, the completion times of the client nodes in the new topology are all around the same time as opposed to the old topology, which has more scattered times. The convergence of times for the clients in the new topology is because since some of the client nodes' latency decreased due to the topology change, all client nodes have a reasonable chance to be first in line to get query the server. That is why the previously closest nodes increased in completion time in the new topology.

3.4 Multi-Server Multi-Client Scenario

Server Node	Old Topology QPH	New Topology QPH
Server 4	333.6	340.2
Server 10	331.2	327

Table 6: Queries per hour at scale factor 0.01 for each server servicing each client under old and new topologies in the multi-server multi-client scenario.

In this scenario, two servers can be considered two separate applications running on all clients that each server is

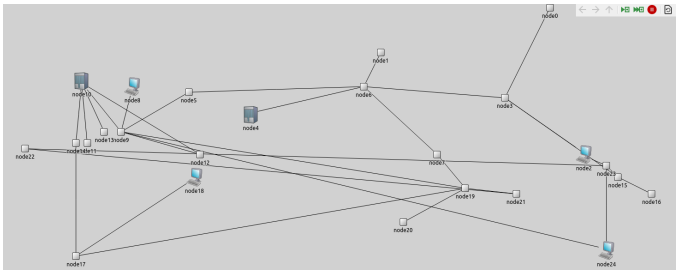

Figure 6: This is the multi-server multi-client original topology where the clients are denoted as computers, and the servers are shown as servers. Client node 8 is located at the top left, client node 18 is at the bottom right, client node 2 is in the middle right, and client node 24 is at the bottom right. Then server node 4 is in the middle, and server node 10 is in the upper left.

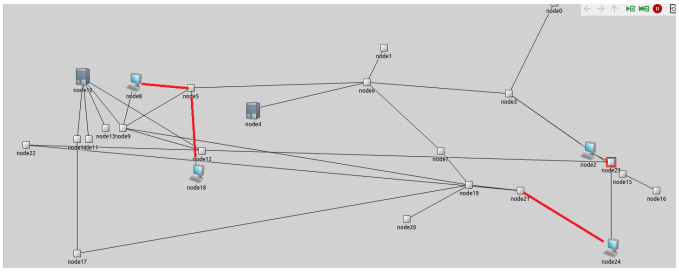

Figure 7: This is the multi-server multi-client new topology where the clients are denoted as computers, and the servers are shown as servers. Notably, the topology changed the same way as in fig 5, and new connections that will be used can be seen in red.

Server	Old Topology (s)	New Topology (s)
Server 4	10.67	10.24
Server 10	10.26	10.29

Table 7: Average time for each server to fulfill the client's queries under old and new topologies in the multi-server multi-client scenario.

trying to service or replicated processes servicing the same clients to increase fault tolerance. Either way, in this scenario, we want to evaluate how changing the topology in a multi-application/multi-server setting can impact the performance metrics of the system(s) as a whole.

For the multi-server multi-client scenario, we consider a 4-client 2-server scenario. The original topology for this scenario builds off the multi-client multi-server original topology by adding a server node randomly assigned a location that turns out to be in the upper left node 10, shown in fig 6.

Client/Servers Nodes	Old Topology (s)	New Topology (s)
Client 2 Server 4	8.01	9.82
Client 8 Server 4	8.85	9.55
Client 18 Server 4	10.56	10.08
Client 24 Server 4	9.03	10.23
Client 2 Server 10	10.22	10.29
Client 8 Server 10	4.98	4.67
Client 18 Server 10	7.61	7.73
Client 24 Server 10	10.1	10.12

Table 8: Average time for each client node to complete its set of queries for each server in the old and new topology in the multi-server multi-client scenario.

For this topology, we simulated to extract metrics for each server's throughput test and queries per hour test. We first conducted 10 throughput tests, which yielded an average time of 10.67s for server 4 and 10.26s for server 10 to process all of the client's queries, as seen in Table 7. Also, notably, all the client's completion times for the throughput test are in Table 8. Then we conducted a simulation to record the queries per hour, which, with a scale factor of 0.01 on the database size, yielded queries per hour of 333.6 on server 4 and 331.2 on server 10, as seen in Table 6.

Having our initial metrics of the system, we can improve the system's performance by performing the same topology changes as the one-server multi-client scenario, as seen in fig 7. Then we use this new topology and extract the same performance metrics again, yielding an average completion time for the throughput tests as 10.24s for server 4 and 10.29s for server 10, as seen in Table 5. Then for queries per hour, server 4 got 340.2, and server 10 got 327, as seen in Table 6.

4 DISCUSSION

5 CONCLUSION