NETWORKING FOR BIG DATA: DATA CENTERS

Challenge #1: Topology Design

2023/2024

QUIMBY

Simay Çalışkan - 2105260

Dila Aslan - 2113310

Umut Altun - 2101934

Mayis Atayev - 2104359

1. ER Graphs versus r-Regular Graphs

Firstly, two random graphs (ER graph and r-Regular graph) have been created. Afterwards, three functions were created to determine whether a graph is connected according to the following algorithms: irreducibility, Laplacian Eigenvalue, and breadth-first search. These three functions have been used to analyze the complexity of the connectivity for different numbers of nodes (*K*).

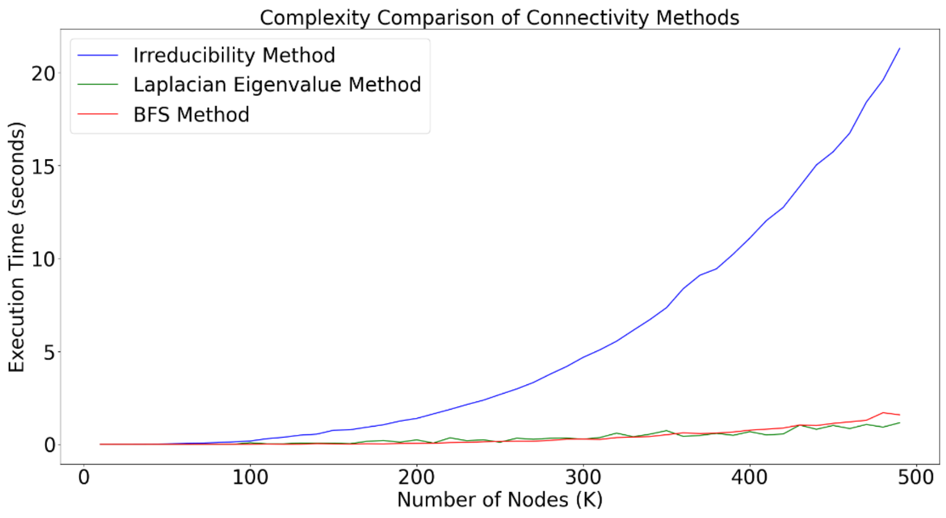
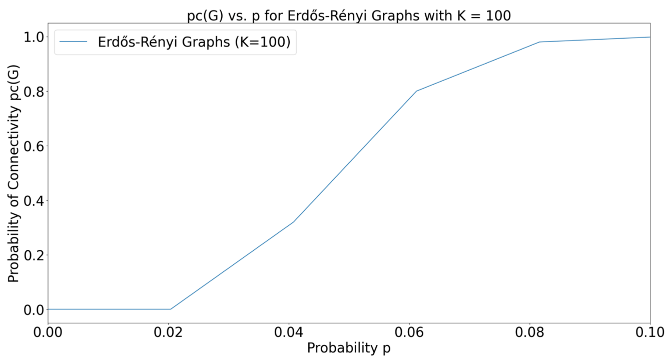


Figure 1: *Complexity versus Number of Nodes Graph*

As can be seen on *Figure 1*, the irreducibility method has a cubic complexity, growing significantly faster than the other methods as K increases. The Laplacian eigenvalue method also shows cubic complexity but appears more efficient than the irreducibility method. On the other hand, the BFS method has a quadratic complexity, scaling better with larger graphs; confirming its practical efficiency for large-scale connectivity checks.



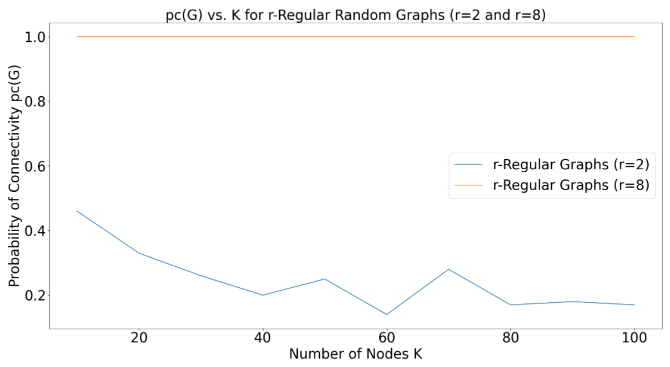


Figure 2: *Probability of Connectivity of ER*

Figure 3: *Probability of Connectivity of r-Regular*

For the ER graph, it can be concluded from *Figure 2* that as the *p* value increases, the probability of the graph being connected rapidly approaches 1. Additionally, there is a critical threshold around $p = 0.08$ where the graph transitions from likely disconnected to likely connected. This aligns with theoretical results which suggest connectivity in Erdős-Rényi graphs increases sharply near *log(K)/K.*

For the r-Regular graphs, two different scenarios for $r = 2$ and $r = 8$ are shown in *Figure 3*. For $r = 2$, the probability of connectivity decreases with increasing number of nodes (*K*), indicating that 2-regular graphs are less likely to remain connected as they grow larger. For $r = 8$, the probability of connectivity remains high and constant, showing that 8-regular graphs are highly likely to stay connected even as the number of nodes increases.

2. Local versus Distributed Run

2.1. Mean Response Time Algorithm

1.Define Hop Count Functions:

Fat-Tree ($h_{ft}(N)$):

First ($n/2$) servers: hop count = 2
Next ($(n/2)^2$) servers : hop count = 4
Remaining servers: hop count = 6

JellyFish ($h_{jf}(N)$):

First ($n/2$) servers: hop count = 2
Next ($(n/2)^2$) servers : hop count = 3
Remaining servers: hop count = 6

2.Calculate Transmission Time:

- Add overhead to data size: ($data' = data + f \cdot data$)
- Inverse RTTs: ($inverse\_T = [\frac{1}{2\tau h_i}]$)
- Throughput: $\theta_i = \frac{c}{\sum(inverse\_T)} \cdot inverse\_T_i$)
- Transmission time: ($T_{trans} = \frac{data'}{\theta_i}$)

3.Calculate Response Time:

- Determine hop counts
- Split input data: ($L_{in} = \frac{L_f}{N}$)
- Generate output data: ($L_{out} = uniform(0, \frac{2L_f}{N})$)
- Calculate job execution time: ($T_{job} = exponential(\frac{E[X]}{N}) + T_0$)
- Calculate transmission times for input and output data
- Total response time: maximum of combined times

4.Monte Carlo Simulation:

- Initialize response time arrays
- For each number of servers (1 to N_{max}):
    - Run M simulations
    - Compute and average response times

5.Run Simulations:

- Fat-Tree: ($fat\_tree\_simulation(N_{max})$)
- Jellyfish: (($jellyfish\_simulation(N_{max})$))

Figure 4: *Algorithm for Mean Response Time*

2.2. Plots

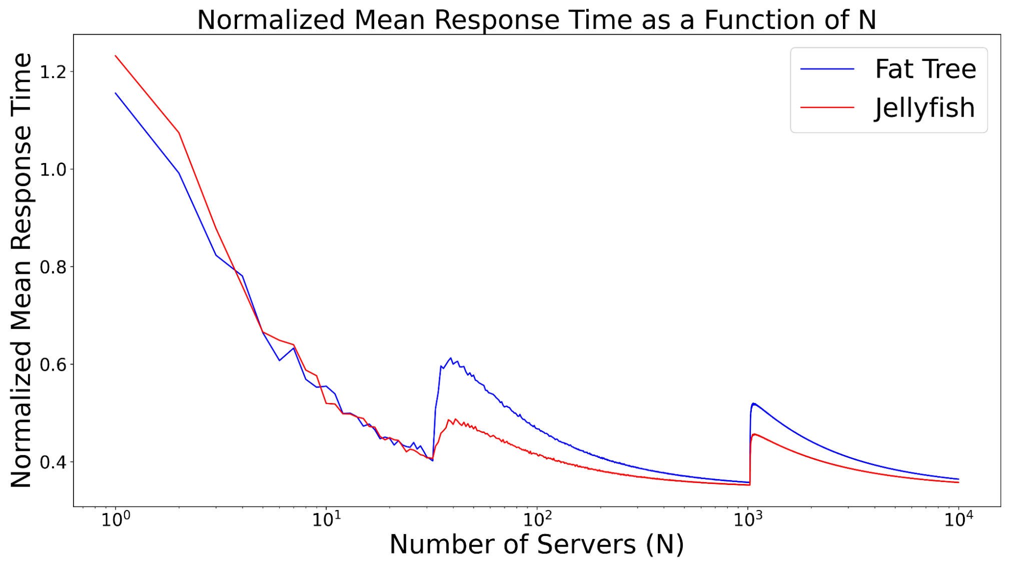


Figure 5: *Plot of the Normalized Mean Response Time*

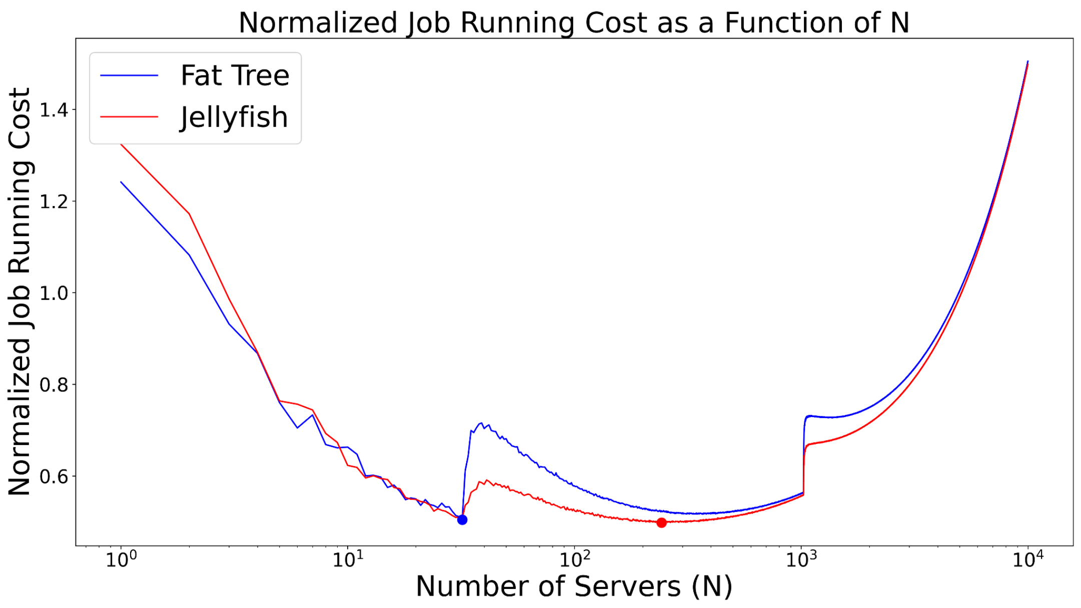


Figure 6: *Plot of the Normalized Job Running Cost*

2.3. Optimal Number of Servers

By determining the points which minimize the graph shown on *Figure 6*, the optimal number of servers for Fat-Tree topology has been concluded as 32 servers; and for the Jellyfish algorithm as 242 servers.

2.4. Analysis of Results

When comparing the mean response times shown on *Figure 5*, both topologies initially show a steep decrease as the number of servers increases, indicating improved efficiency with more servers. However, beyond a certain point, Jellyfish consistently outperforms Fat Tree, maintaining lower response times due to its more flexible and efficient routing, which reduces congestion and improves data transmission efficiency.

As can be seen on *Figure 6*, at the beginning, the job running cost decreases as the number of servers increases, showing efficiency gains. However, as the number of servers continues to grow, the cost for the Fat Tree topology increases sharply beyond a certain point, while the Jellyfish topology remains more stable and cost-effective. This increased efficiency in the Jellyfish topology can be attributed to its ability to distribute network traffic more evenly, reducing congestion and bottlenecks that typically plague Fat Tree structures. Additionally, the algorithm's Monte Carlo simulations likely capture the variability and robustness of the Jellyfish topology, resulting in lower and more consistent job running costs across varying numbers of servers. This makes Jellyfish a preferable choice for scalable data center networks where cost efficiency and performance stability are crucial.