SAPIENZA
UNIVERSITÀ DI ROMA

I3S DEPARTMENT
MASTER DATA SCIENCE

CHALLENGE 1

NETWORKING FOR BIG DATA AND LABORATORY

Professors:
Andrea Baiocchi

Gray Group:
Giuseppe Di Poce 2072371
Mario Edoardo Pandolfo 1835189
Gabriele Pelliccioni 1838084

Contents

Academic Year 2022/2023

1 Complexity Analysis on Graphs Connectivity

Time Complexity: As the plot shows, the ranking order of the three methods is independent from the type of graph. It is clear that *Irreducibility* is the slowest method (we calculate $n-1$ powers of the matrix A) and it seems to have a *polinomial* time complexity of $\mathrm{O}(K^i)$ with $i \in \mathbb{N}$, and, in this case, we can see that it is approximable to K^4.

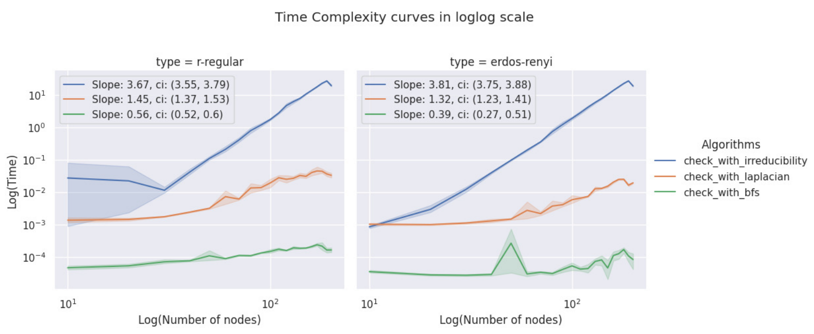

Figure 1: Generated with $K = 200$ and m=10 (simulation size).

We know, theoretically, that *BFS* has a *linear* behaviour due to the fact that increase by $O(K + E)$. The *Laplacian* method is in the middle of these two in terms of time complexity, with a slope bigger than 1 we can suppose a polinomial behaviour (more in space complexity section). It is evident how the three methods work on different orders of magnitude in terms of time complexity, with the *BFS* method performing the best. Additionally, because *BFS* algorithm is also affected by the number of edges, the curves in the plots (for some values of K and p) seem to follow different time complexities, this tho depends from the fact that in the *ER* graph the E number is a random variable which $E[\#edges] = \frac{K(K-1)}{2} \sim O(K^2)$.

Space Complexity: To measure the space complexity between the *Laplacian* and the *BFS* approach we have used the pip library 'memory-profiler' (to run, from the directory where there is the script: 'mprof run script.py'). In the same conditions, the *BFS* took 22 seconds and used 165 MiB, while the *Laplacian* took 13 minutes and used 3700 MiB of memory (this tells us that the *Laplian* has not a linear time complexity). This result does not surprise us because *BFS* has to take in memory only the visited nodes (space order of $O(K)$), while both *Laplacian* and *Irreducibility* have to at least take in memory the whole adjacency matrix ($O(K^2)$).

Conclusion: *BFS* is the best algorithm for checking graph connectivity for both types of complexity.

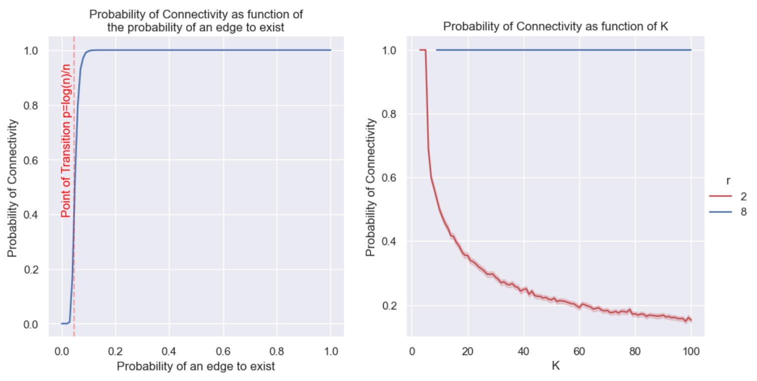

Figure 2: p_c behaviour for Erdos-Renyi (left) and RRG (right) with M=1000 (sim. size)

Probability of Connectivity: In the *ER* graph plot the p_c has a step-behaviour for which $p_c \approx 0$ for $p < \frac{\log K}{K}$ and $p_c \approx 1$ for $p > \frac{\log K}{K}$, this is an important characteristic of the *ER* graphs and takes the name of *Phase Transition*. In the *RRG* plot instead, it is shown that if $r = 8$ the graph has $p_c = 1 \quad \forall k \in K$, while for $r = 2$ the p_c trend is decreasing as the number of nodes increases.

2 Response Time Evaluation

The following is a statement in pseudo-code of algorithm for the evaluation of the mean response time:

Algorithm 1 Response Time Evaluation Algorithm

```
1: simulation ← Array of zeros of dimension $n \times 1$
2: for $m$ in SimulationSize do
3:     $G$ ← initialize graph
4:     A ← choose randomly a server from G
5:
6:     response_time ← Empty dictionary
7:     for $n$ in N do
8:         hopes$_i$ ← G.get closest servers distances
9:         $X_i$ ← from random exponential distribution of average $\frac{E[x]}{n}$ take $n$ samples
10:        $Lo_i$ ← from random uniform distribution among $[0, 2 \cdot \frac{Lo}{N}]$
11:        $T_i$ ← $2 \cdot \tau \cdot$ hopes$_i$
12:        average_throughput_i ← $\left(C \cdot \frac{1}{T_i}\right)$ / MEAN($T_i$)
13:        time_in_going ← $\left(\left(\frac{L_f}{n}\right)(1+f)\right)$ / average_throughput_i
14:        time_in_returning ← ($Lo_i \cdot (1+f)$) / average_throughput_i
15:        working_time ← $T_0$ + $X_i$
16:        response_time[$n$] ← MAX(time_in_going + working_time + time_in_returning)/($T_0$ + $E[X]$)
17:
18:    end for
19:    simulation += response_time
20: end for
21:
22: Estimate_R ← simulation / SimulationSize
23: done
```

Where:

→ G is the graph of the Data Center topology;

→ A is the server chosen randomly from which we compute the response time (RT);

→ hopes$_i$ are defined as the number of links a data packets have to pass through to reach destination i. The number of hopes required to transmit data between two points within a data center can impact the performance and latency of the network. In general, the fewer hopes required, the faster and more efficient the communication will be. Notice that in this case hopes$_i$ is an array of dimension $n \times 1$;

Notes:

Being *Fat-Tree* a deterministic and regular topology, to reduce the time required for our simulations, we have initialized G and chose A outside the simulation loop.

The pseudo-code does not take in consideration that the bootstrap procedure is done automatically by the 'seaborn' library with which we're doing the plots: instead of adding directly the response_time vectors to the simulation vector we save each result in a pandas.DataFrame and then pass this dataframe to seaborn which will plot a bootstrapped estimate of the curve.

3 Response Time and Job Running Cost

Response Time:
From the response time plot we can see that, while for the Baseline the response time remains constant $\forall n \in N$, for both the *Fat-Tree* and *Jellyfish* topologies we have that it decreases as the number of servers increases. Since the plot is in loglog scale, it is possible to understand that both topologies follow a $O(N^{-1})$ behaviour. Indeed, relationships of the form $y = ax^k$ appear as straight lines, with k as slope, in a loglog graph. This result does not surprise us because in performing a task in a Data Center (DC) we distribute it among N servers, dividing the job by N.

It seems also that the regularity of both graph topologies is able to handle the influence of the hopes to reach the servers: in theory, the more servers we use, the more the length of the paths could increase creating delays.

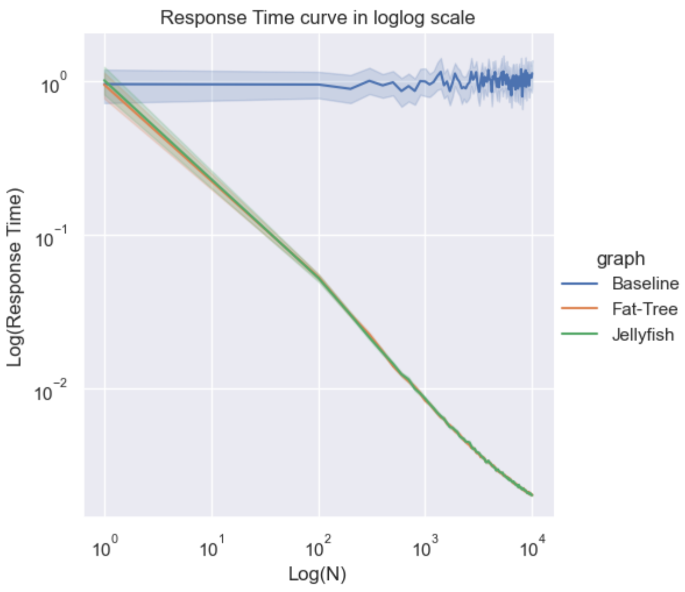

Figure 3: Generated with M=100 (sim. size).

Another aspect to notice is that, while for the Baseline the response time maintains a lot of variance $\forall n \in N$, for both topologies (*Fat-Tree* and *Jellyfish*) it seems to decrease as the used servers increase. This is because, being the time of server usage defined as $T_0 + \frac{E[X]}{N}$, then have that $\lim_{N\to\infty} T_0 + \frac{E[X]}{N} = T_0$, so by adding more servers we're able to reduce the impact of $E[X]$ (the largest delay) on the overall response time. This is true but at the cost of increasing the network workload.

Job Running Cost:
The plot shows that the job running cost stays constant (high variability) for the Baseline, while it has the same behavior for both *Fat-Tree* and *Jellyfish*: it rapidly reach a minimum $\hat{s}$ and then it grows linearly as we add more servers. This behavior is easily explainable if we relax the definition without considering the randomness behind X_i:

$$E[S] \approx T_0 + E[X] + \xi \sum_{n=1}^{N} \left(T_0 + \frac{E[X]}{N} \right)$$
$$= T_0 + E[X] + \xi(N \cdot T_0 + E[X]) \sim O(N)$$

By running different bootstrapped simulations with size 1000 we have found that both topologies have the minimum $\hat{s}$ for an $N \in [200, 240]$ (with *Jellyfish* having it before than *Fat-Tree*), but the variability of the process makes finding a clear minimum very difficult even for simulations of size 1000.

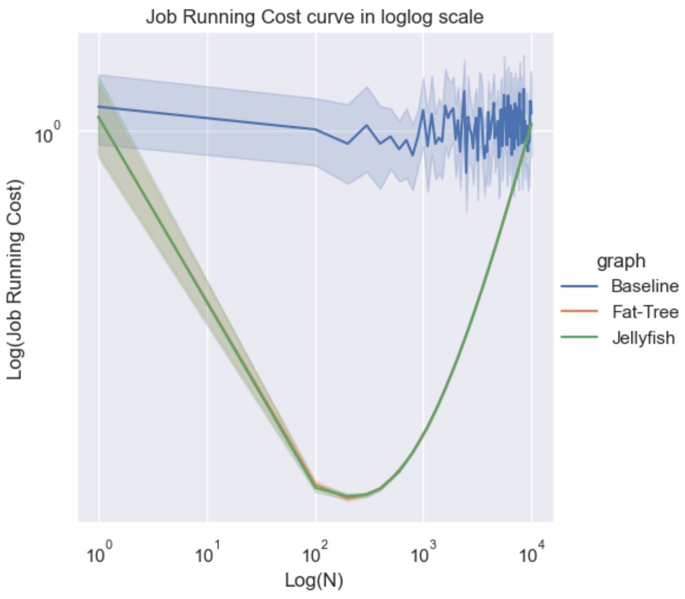

Figure 4: Generated with M=100 (sim. size).

Conclusions: There are no appreciable differences between the two topologies regarding the response time and the job running cost.