NETWORKING FOR BIG DATA

Group Rivest

Author:
Bassani Aurora, 1852791
Luciani Erica, 1868647
Mignella Laura, 1920520

1 Part 1

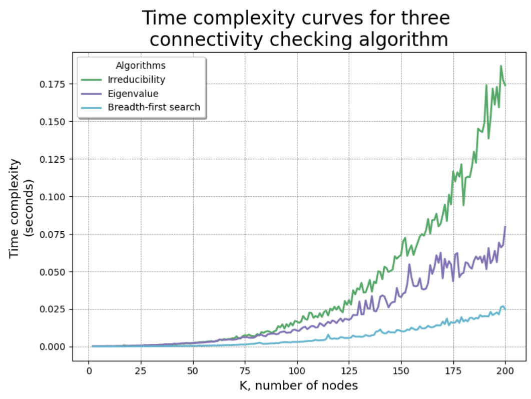

Figure 1

In **Fig. 1** it's represented the complexity time of the three connectivity checking algorithms tested on Erdős–Rényi graphs with probability p that two nodes are linked equal to 0.5. We can notice, as we expected, that as the number of nodes K grows the time complexity of each method increases. In particular, the Bread-first search is the fastest one when K is large and the irreducibility method is the worst one. Nevertheless, we tested the algorithms using different probabilities p and the result was that the behavior of the BFS changed significantly. Namely, for bigger p, when K grows the BFS gets worse and the best algorithm becomes the eigenvalue. This change is due to the fact that if p is big the number of edges in the generated graph is presumably bigger.

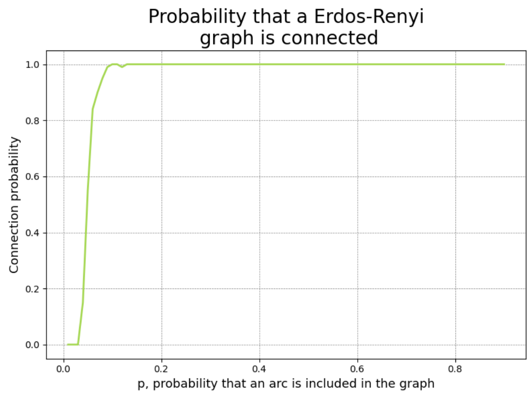

Figure 2

In **Fig. 2** you can see the evolution of the probability that an Erdős–Rényi graph is connected over p considering a fixed number of nodes K equal to 100. To obtain this result BFS was used since from previous tests we've seen that with 100 nodes the method has still a low time complexity even if p is large. No surprise here, we expected that the computed probability would soon reach one. This is due to the fact that each node has a probability p to be linked to each other node and since the number of nodes is not too small the probability that it is connected to at least another node is soon high.

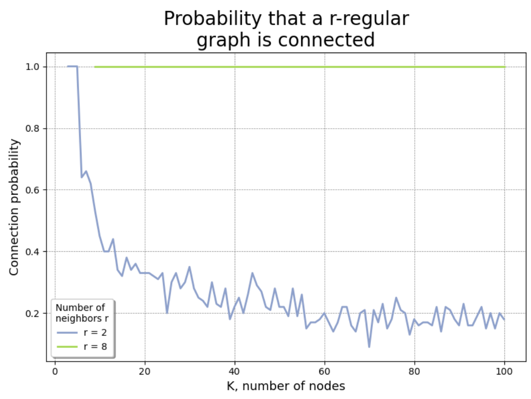

Figure 3

In **Fig. 3** you can see the evolution of the probability that an r-regular graph is connected over K considering two degree centrality, namely $r = 2$, $r = 8$. We can notice that if $r = 8$ the computed probability is always 1 since we are studying its evolution for $9 \leq K \leq 100$ and each node is linked to exactly 8 nodes. On the other hand with $r = 2$ the probability decreases as K increases since it becomes more probable that there are cycles disconnected from the rest of the graph.

2 Part 2

Given a topology -fat tree of jellyfish-, the number N of servers that will share the work, a dictionary with the attribute of each node -'host' or 'switch'- and a list containing all the servers, we first randomly choose Server A among all the servers and compute the size of input data for each of the servers. Then we compute the shortest path between Server A and all the other servers and take only the N nearest to A. We store in a list the number of hops needed to go from Server A to each of the N nearest servers and then we use the information we gained to compute the average throughput. We find the times (s_t) needed to transfer data from server A to each of the N servers as the ratios of input data over the average throughput. At this point we generate N samples X_i from an exponential distribution with mean $\frac{E[X]}{N}$, since each server takes $T_0 + X_i$ seconds to have the job done.
Now we need to find the response times that we compute as the ratio of output data size, that we obtain extracting N times from a uniform distribution on $[0, \frac{2L_o}{N}]$, over the average throughput. Finally, we sum the sending times, the times needed to complete the task and the response times, and take the maximum of the sums and divide it by $R_{baseline}$.

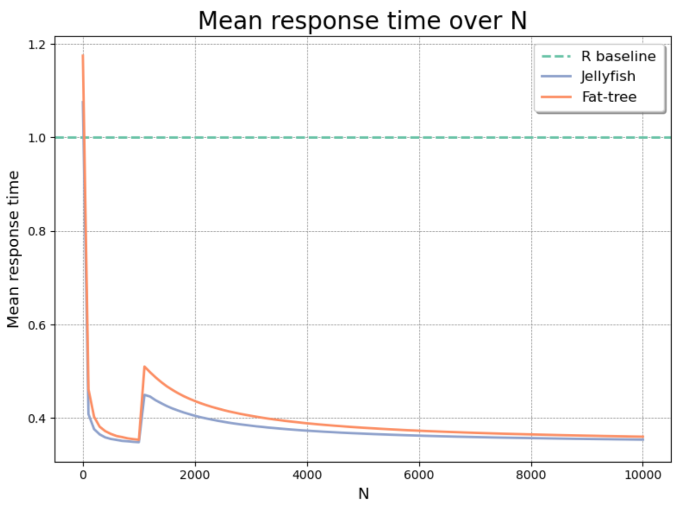

Figure 4

In **Fig. 4** it's represented the mean response time $E[R]$, normalize with respect to $R_{baseline}$, as a function of the number of servers among which the job is divided. As we can see as N increases the average time taken by N servers to perform a certain job is less than the time taken by a single server to perform the same job. Moreover we can also notice that the behavior of the two topologies, jellyfish and fat tree, with respect to the mean response time is almost the same.

Let's zoom some interesting areas of our **Fig. 4**.

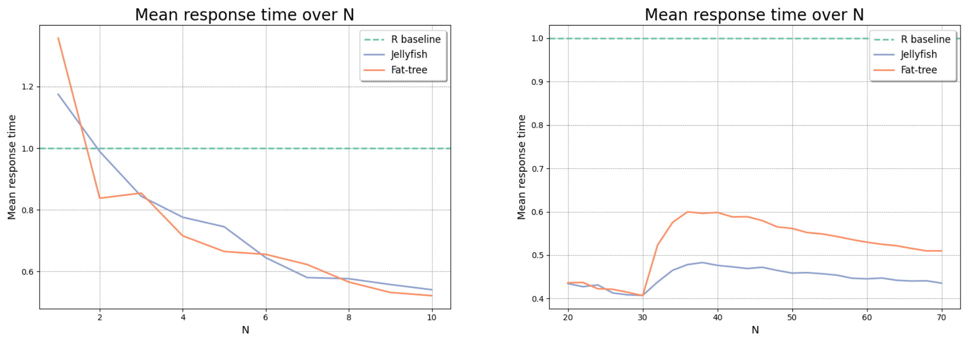

Figure 5

In the image on the left we plotted the mean response time from $N = 1$ to $N = 10$ with a step equal to 1 to check to which N corresponds the first mean response time under $R_{baseline}$. As we can see from $N = 2$ onwards, this condition is satisfied for both the topologies and so it always seems convenient to split the job among different servers.
From **Fig. 4** we can notice that, for both the topologies, around $N = 1000$ there is a peak. This behavior is due to the fact that after some N around 1000 the number of hops between $server_A$ and these N servers goes from 4 to 6 for the fat tree topology and from 3 to 4 for the jellyfish topology, hence the mean response time grows. After the peak, the curve decreases again since the number of hops doesn't change while the number of servers increases. We asked ourselves if this kind of conduct also appears when we pass from 2 to 4 hops in the fat tree topology and from 2 to 3 in the jellyfish topology, which happens when N is over 31.
In the rightmost figure in **Fig. 5** you can find the mean response time for $N \in [20, 70]$ with step equal to 2. As we expected there is a slight peak also in this case.

In **Fig. 6** it's represented the job running cost $E[S]$, normalize with respect to $S_{baseline}$. Since the job running cost depends also on the mean response time, in the plot we can notice a similar evolution at the beginning of the curve and the presence of a kind of a peak around $N = 1000$. When N is over 1000 the curve has a different behavior and this is caused mainly by $E[\Theta]$, the average servers time used to run the job.
In particular, after $N = 1000$ the job running cost starts to grow with N. Furthermore for N over 6000 this cost exceeds $S_{baseline}$ that corresponds to the job running cost of a single server.
This highlights the fact that from some point on the total time required by the servers becomes too much and so, in general, for N big enough it's not always worth splitting the job.
In conclusion, we can say that intuitively the N that minimizes the job running cost is 31 since it is the value where we have the first local minimum and the value before the first peak.

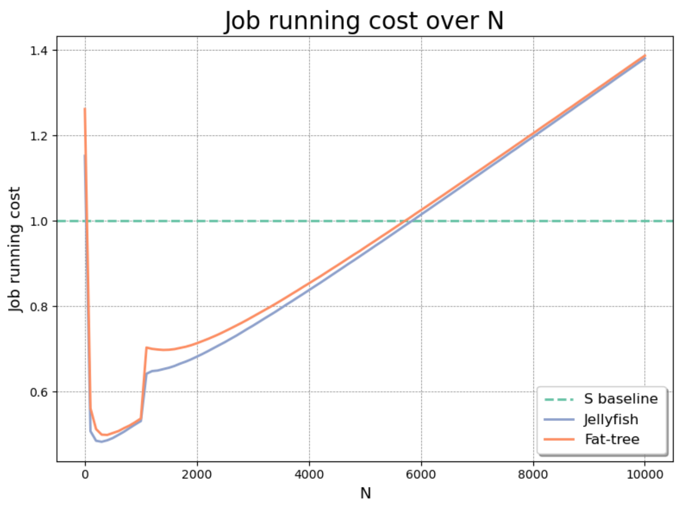

Figure 6