Sapienza
Università di Roma

Department of Data Science (DIAG)

Dijkstra

CHALLENGE 1

Networking for Big Data - Data Centers

Students:

Tito Tamburini
1837335

Angelo Mandara
2077139

Claudiu Gheorghiu
1845227

Professor:
Andrea Baiocchi

Academic Year 2022/2023

The first part of the study focuses on the connectivity of random graphs, in particular p-ER and r-regular ones. We examined *irreducibility* and *eigenvalue of the laplacian matrix*, two algebraic methods, then the *BFS (breadth-first search)* algorithm. In figure 1 it's clear that the first is also the worst. It requires computing the matrix exponential, which has a complexity of $O(K^3)$, where K is the number of nodes in the graph. The second involves computing the eigenvalues of the laplacian of the graph. The complexity is $O(K^3)$, but it can be faster for large graphs via parallelization with linear algebra libraries. The best is the BFS algorithm, with a linear time complexity $(O(K + E))$. To drive our study we fixed $p = 0.5$, $r = 5$. An increase of the number of nodes will raise the sparsity of the graph. The difference between p-ER and r-regular graphs does not affect significantly the result.

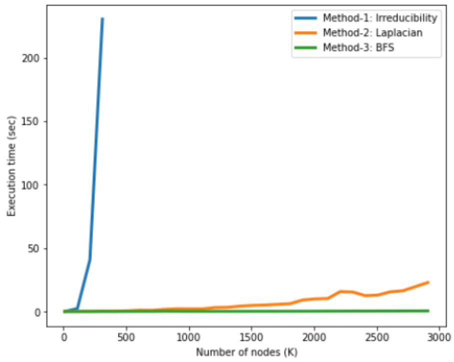


Figure 1: Complexity vs number of nodes K in case of r-regular graph

To estimate the probability of a p-ER graph being *connected*, i.e. that exists a path between any two vertices in the graph, we generated a large number of random graphs, for several values of p and K, and we used the *BFS* method computed before to count how many times the graph generated was connected. For Erdos-Renyi graphs with $K = 100$, in figure 2 we can see that the curve behaves as a step function with respect to the probability of each edge being present. When p is small (≤ 0.03) , the graph is very likely to be disconnected. As p increases, from 0.03 to 0.12, the graph becomes more likely to be connected, and the probability quickly jumps up to 1.

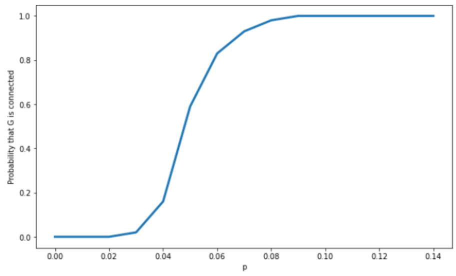


Figure 2: P(connected ER graph) as function of p for K=100 nodes

In figure 3, in the case of r-regular with $r = 2$, as the number of nodes increase, the probability of the graph to be connected decrease. Each node is connected to only two other nodes, which means there is a higher chance of creating isolated groups. With $r = 8$ we can see that the graph is always connected, each node is connected to eight other nodes, which increases the chance of creating connections. It is palpable that the *six degrees of separation* hypothesis has some validity both analytically and in real-world situations.

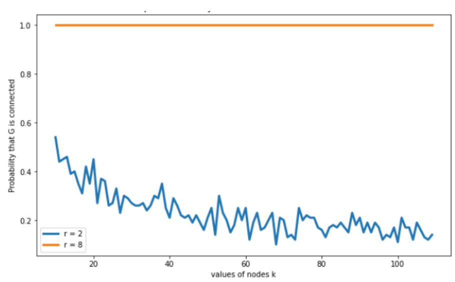


Figure 3: P(connected r-regular graph) as function of K for r=2,8

We want to simulate in a virtual Data Center topology the procedure that server A has to initiate to complete a generic job by splitting the computation to N other servers.
It follows the algorithm we implemented in Python with *Networkx* module to complete the task:

1. **Build a virtual Data Center Topology**:
   Two types of topologies: Fat-Tree and Jellyfish. Each switch must have $n = 64$ ports.

   (a) *Fat-Tree*:
   - calculate the number of core switches ($(\frac{n}{2})^2$), aggregator switches ($\frac{n}{2}$ for pod), edge switches ($\frac{n}{2}$ for pod) and servers needed ($\frac{n}{2}$ for edge switches, so in total $\frac{n^3}{4}$);
   - create an empty graph and carefully populate it with nodes of different types: first core, then aggregation and edge switches;
   - add the edges to connect the switches together respecting the structure of the fat tree topology and finally to each edge switch add $\frac{n}{2}$ servers.

   (b) *Jellyfish* :
   - same number of switches as the Fat-Tree, with $r = \frac{n}{2} = 32$ connections between each Tor switch. The function $r - regulargraph(K, r)$, returns a graph where each node has degree r. For each node we hold the attribute *Type* = *switch* because are added $n - r$ servers to each switch; after that the Jellyfish Topology is completed.

2. **Simulation of response time R and job running cost S in function of the servers' number N used to split a generic job received by a server A in the given topology:**

   (a) Let A be a randomly chosen server from the given virtual network.

   (b) Find the N nearest servers to A in the topology, using an implementation of Dijkstra's algorithm in $Networkx$;

   (c) For each i-th neighbor, with $i = 1, \ldots, N$, take a sample X_i from an exponential distribution with mean $E[X]/N$ to simulate the time it would take to complete a task of the job, where X is a random variable that represents the time it would take to complete the job if only A was used. Here, $E[X] = 8$ hours $= 8 \times 3600$ seconds.

   (d) For each i-th neighbor, with $i = 1, \ldots, N$, take a sample from a $Unif[0, 2\frac{L_o}{N}]$, where L_o is the dimension of the output if only A worked on the job. Here, $L_o = 4$ TB $= 4 \times 10^{12}$ bits. This simulates the i-th output $L_{o,i}$ computed by the i-th neighbor server.

   (e) For each i-th neighbor, with $i = 1, \ldots, N$, compute the RTT t_i between itself and A, given by the formula $t_i = 2\tau h_i$, where h_i is the distance between the i-th neighbor and A in the unweighted graph implementation of the topology.

   (f) Given a vector with all the t_i's, compute the average throughput $\theta_i = \frac{1/t_i}{\sum_{j=1}^{N} \frac{1}{t_j}} \times C$, which gives the mean number of bits/s flowing in the connection between the i-th neighbor and A, where $C = 10^9$ bits/s is the capacity of each link in the topology.

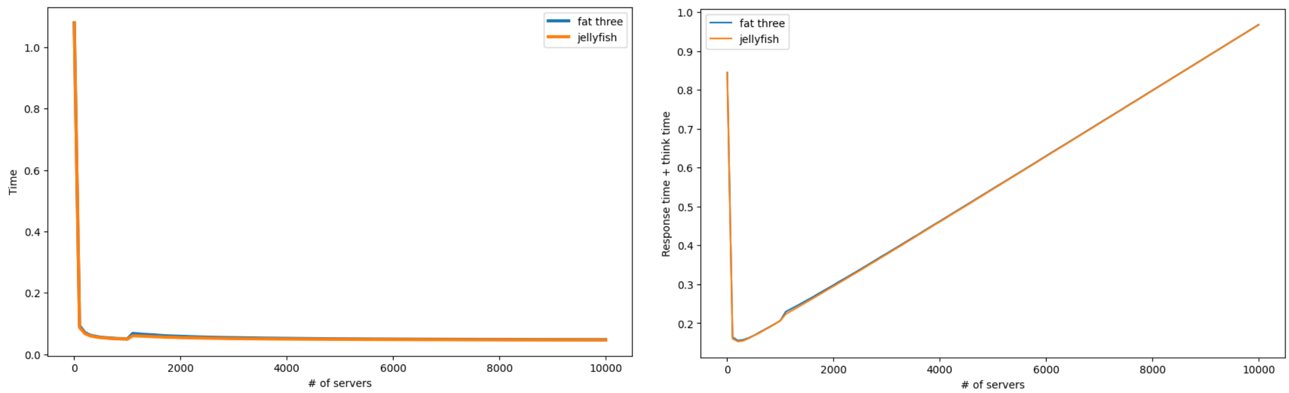
(g) Compute, for each neighbor, the time it takes to receive the input L_f/N from A and send the output $L_{o,i}$ to $A : T_i = \frac{(L_f/N + L_{o,i})(1+f)}{\theta_i}$; where $f = 48/1500$ is the TCP overhead added when transmitting the data, a fraction of the orginal data.

(h) Finally, can be derived both the *response time* $R = T_0 + \max(X_i + T_i)$ and the *job running cost* $S = \sum_{i=1}^{N}(T_0 + X_i + T_i)$; where X is the vector containing all the Xi's, same for T with the Ti's.

3. **Repeat 100 times step 2 for N going from 1 to 10000:**

(a) By repeating for each N 100 times the simulation it can be computed the mean of each value of R and consequently S. Notice that in the case of Jellyfish topology also the choice of the server A have to be randomized, since the last has an aleatory interconnect scheme. Fat tree, in contrast, has a deterministic structure and does not depend on the choice of A.

(b) So we can be plot (figure 4) the average expected time R (4.a) and the job running cost S (4.b) depending on the number of servers N used to split the job, normalizing over the value of R_{base} and S_{base}, which are the response time and job running cost, respectively, in case only server A is used.



(a) Mean response time E[R]

(b) Job running cost S

Figure 4: Fat Three vs Jellyfish topology

We can observe that the response times for the two topologies are similar: both attempt to provide high-speed communication between switches and servers while offering a balanced distribution of network traffic. In general, it is better to distribute the workload across more servers, but only up to a point (as shown by the plot of the Job running cost S), after which it will reduce performance. The performance will be significantly reduced by adding more servers as the network traffic and communication overhead will increase (in our code, X_i will be smaller but we will keep adding T_0). The number of server for which we obtain the minimum value of Job Running cost S is 192 for both topologies. In conclusion, it is critical to do a performance study to find the ideal number of servers and prevent overloading the network.