Scalable distributed topologies

Content

1) Notions

A simple graph is an undirected graph with no loops and no more than one edge between two vertices.

A complete graph is one that each pair of vertices has an edge connected them (the star graph)

A connected graph is one that there’s a path between every two nodes.

A star graph is the one that has a central vertice and many leaf nodes connected to the center.

A tree is a graph with no cycles.

A planar graph is a graph that can be drawn without two edges intercepting each other.

Degree: number of adjavent vertices to vi. In directed graphs we also have the in-degree and the out-degree.

The **distance** $d(v_i, v_j)$ between two vertices is the length of the shortest path between these.

The **eccentricity** is the **maximum distance** that a vertice might have between every point in the graph ⇒ $\max(\{d(v_i, v_j)|v_j \in V\})$

The **diameter,** on the other hand, is the maximum eccentricity that can be found in the graph ⇒ $\max(\{\texttt{ecc}(v_i)|v_i \in V\})$

The **radius** is the minimum eccentricity ⇒ $\min(\{\texttt{ecc}(v_i)|v_i \in V\})$

The **center** is a node that has the eccentricity equals to the **radius ⇒** $\{v_i|\texttt{ecc}(v_i) == R\}$

The **periphery** is a node that has the maximum **eccentricity** in the graph. $\{v_i|\texttt{ecc}(v_i) == D\}$

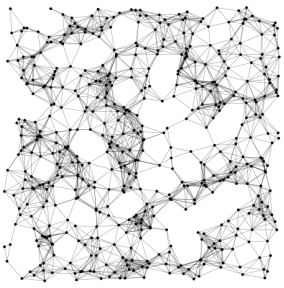
In a **walk** edges and vertices can be repeated.

In a **trail** only vertices can be repeated

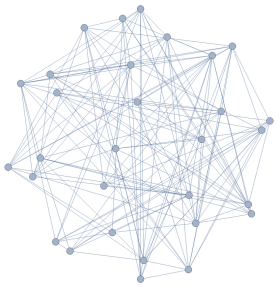
In a **path** no edges and no vertices are repeated.

2) Graph types

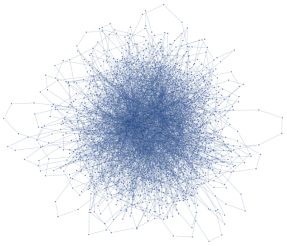
- **Random Geometric:** we drop nodes into a unit square graph and connect nodes that are within a certain euclidean distance range.



- **Random Erdos-Renyi: n** nodes are connected and the probability of an edge be built is **p** and is independent of other factors.



- **Watts-Strogatz model**
- **Barabasi-Albert** the more connected is a node, the more likely it is to receive new links. *this model describes a series of human relations such as social network, internet and others.*



3) Spanning trees (sync)

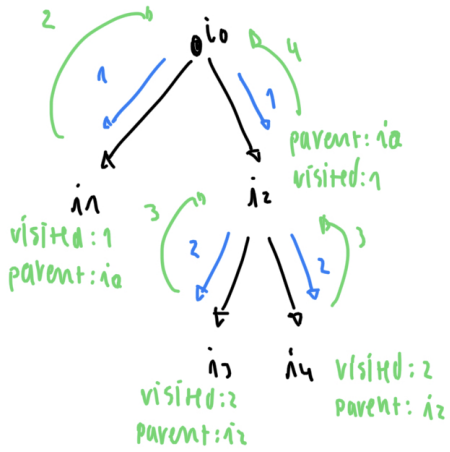
A **strongly connected graph** is one that for every pair of nodes (v,u) there's a path from v to u and a path from u to v. Every strongly connected graph has a breadth-first directed spanning tree and the distances from the root **r** to another node **v** is actually the depth of this node is the graph.

SyncBFS

How can we synchronize the nodes of a Spanning Tree?

- All the nodes starts with the state **visited** = false and with **no parent.**
- **algorithm:** The node i_0 sends messages to its children and the children do the same. When a node receives a message they are marked as visited and set the parent. Once

marked as visited the node can't receive messages of the same type.



- **time:** The time depends on e_0 eccentricity. This eccentricity can be minimum (radius) and maximum (diameter). So the time complexity varies between $[r, d]$.
- **termination:** How to know if the process has finished? A node just finishes when all children have finished. On this way the root can know interactively if the process has ended.
- **applications:** we can make aggregations; use to make leader election, where the larges UID wins; we can also use to make broadcast (a message payload m can be attached to SyncBFS construction or broadcasted once the tree is formed). We can also use it to compute the diameter.

Async spanning

Algorithm

Desempenha duas operações atomicas:

- Se foi feito um `send("search")i,j` por um nó qualquer **i**, a pré condição para isto acontecer é que haja uma variável `sento(j)` devia estar a `True`. Quando a mensagem é finalmente enviada esta variável é posta a `False`

e o nó `j` irá receber `receive("search")j,i`.

Signature:
Input: $receive("search")_{i,j}$
Output: $send("search")_{i,j}$
Transitions:
$send("search")_{i,j}$
Precondition: $sendto(j) = yes$
Effect: $sendto(j) := no$
$receive("search")_{j,i}$
Effect:
if $i \neq i_0$ and $parent = null$ then
$parent := j$
for all $k \in nbrs \setminus \{j\}$ do
$sendto(k) := yes$

- O efeito deste envio no nó `j` é: se `j` não possui `parent` e se não é `i0` (nó inicial), então `j` , assim como na abordagem síncrona, tem de enviar uma mensagem a todos os seus nós filhos. Ou seja, colocamos `sendto(k) = True` para todos os vizinhos de `j`.

Not a BFS

Não necessariamente a **async spanning tree** vai produzir uma **BFS,** porque o caminho escolhido nem sempre vai ser o mais curto, mas simplesmente o mais rápido.

Se o tempo para processar a mensagem é `l` e o tempo para mensagem ser entregue é `d`, então temos que a complexidade temporal **não** é `O(diam(l+d))` , porque nem sempre teremos o menor caminho entre dois nós. A complexidade temporal pode ser `O(h(l+d))`, onde `h` não pode demorar mais do que o tempo se `h=diam` , já que estamos a procurar a path mais rápida. Ainda, é possível quem alguns casos `h=n` .

Broadcast with acks

This broadcast system is similar to the sync option. When the child receives the message, they send an **ack** to the parent and the parent just acks, once all the neighbors responds with an ack. If a node has already received the message it also responds with an ack.

This same system is useful to make election systems.

4) Flooding vs Broadcast

Flooding is a simple algorithm that sends the same packet for all edges of a node. The difference is that broadcast implies that all hosts will receive the package once, while in flooding the same package may be receive more than once in different times.

5) Epidemic Broadcast Trees

Gossip broadcast

https://www.gsd.inesc-id.pt/~ler/reports/LPR_GossipBasedBroadcast.pdf

The gossip broadcast is similar to the flooding algorithm: we select **f** direct to nodes to send the message (in flooding we usually would select all the neighbors). The gossip broadcast allows the same node to receive the message **twice** and when this happens the message is discarded.

- [**+] this system is more resilient to failure.** If a communication with another node fails it also may be contacted by other ways.
- [**-] does't optimize the network traffic.** There's a excessive message overhead.

Tree-based broadcast

This broadcast is just like we have seen in the previous section.

- [**+] small number of messages in the network**
- [**-] fragile in the presence of failure**

Epidemic broadcast

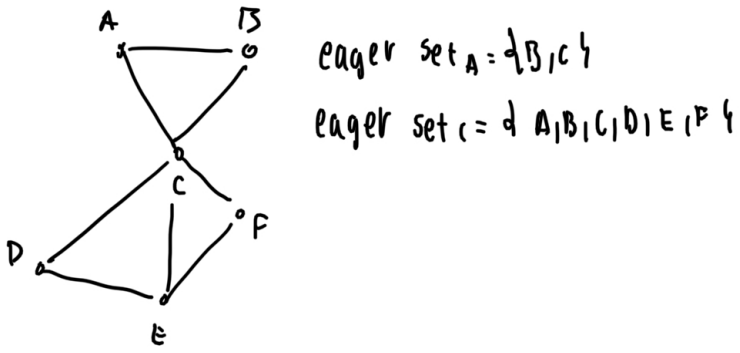
https://www.gsd.inesc-id.pt/~ler/reports/srds07.pdf

Some concepts

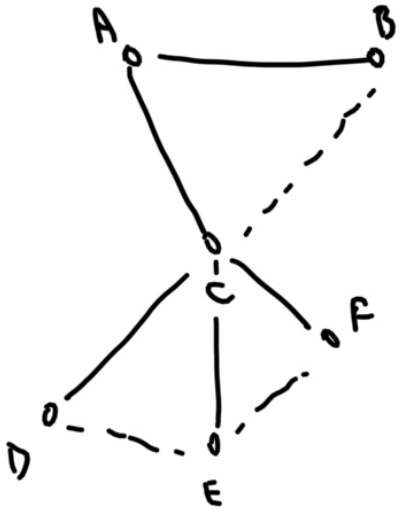
- **Eager push -** The translation to “eager” is “ansioso”. Which means that barely the message is received it’s forward right away.
- **Pull -** Periodically the nodes asks if there’s new messages.
- **Lazy push -** When a peer receives a message for the first time, it selects random peer and send the **ID** of the message (not the payload). If the peers didn’t receive the message yet, they may request a **pull**. **There’s a separation between payload and metadata.**

Gossip into tree

- Each node will choose random nodes to which they will connect. By the end of this process we gonna have a network. **How do they do that?** *They can use something called peer sampling service: initialize a random walk in the network and after x steps it is reported the network state.*
- Each node selected will have the **eagerPush set.** For example:



- The first message reception puts the origin as **eagerPush.** So, if A sends a message to B, then it will be set as **eagerPush.**
- Duplicated messages will set the sender as **lazyPush.** For example, let’s suppose that A sent a message to C and then B sends the same message to C. When C receives the duplicated message from B it will set it in **lazyPush.**



The weak connections have the lazy push. However if the network fails, in contrary of what happens in the **tree-based broadcast,** the weaker connections may be used for recovering.

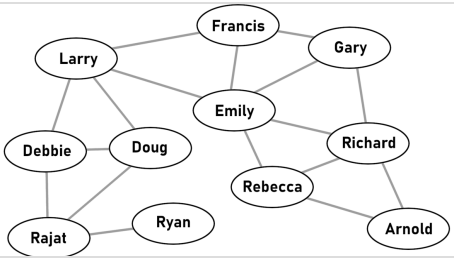
- A connection break may be detected by a timer (receives the metada before the payload) and in this case the **metadata source in lazyPush is set to eagerPush.** If there's a redundancy in the network, the node will receive two messages and by the duplicate reception, the second node that sent the message is set as **lazyPush.**

6) Small Worlds

Six degrees of separation - Wikipedia

Six degrees of separation is the idea that all people are six or fewer social connections away from each other. As a result, a chain of " friend of a friend" statements can be made to connect any two

https://en.wikipedia.org/wiki/Six_degrees_of_separation



The theory that each people in the world is separated by 6 connections.

Random graphs and clustering

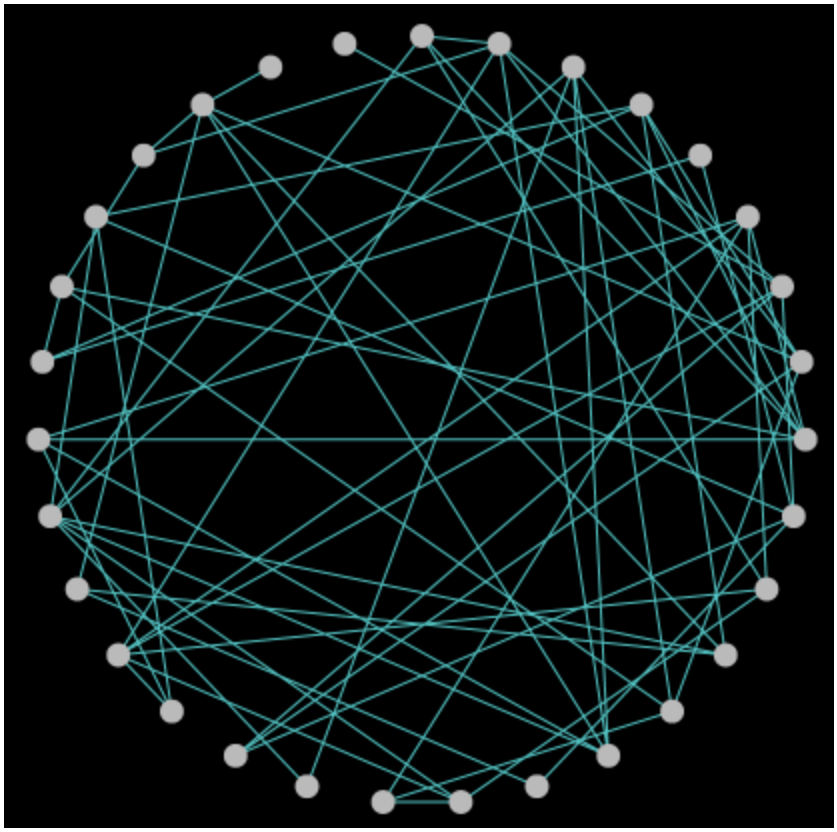
The graph of social relations doesn’t seem like Erods-Renyi. It looks more like Baribasi-Albert.

> The Erdos-Renyi graph does not contains two very important properties:
> - Doesn’t generate a local aglomeration, because it has a constant probability of two nodes being connected.
> - Doesn’t count to produce hubs.

Watts and Strogatz proposed a model that mixes short range and long range contacts. It’s something that remembers chord (short contacts are the adjacent nodes in a ring).

This results in a **low diameter** and **high clustering.**

The point is that imagine that we have to get a route between two points x and y. To find a path between these two points may not be so easy to find:



Wattz-Strogatz, p=1.

The image above, two close points in the biggest part of the time doesn’t have a link between then, which makes the communication complicated. **There is a lack of locality.** For this reason **Kleinberg** solved this issue by choosing a probability function that can restore the locality along the links. **Chord** has this property of uniformity across long distances.

Questions

1) What is a simple graph?

2) What is eccentricity?

3) What is the relation between radius, diameter and eccentricity?

4) The internet is similar to which graph model?

5) What are the graphs similar to social interactions? Why?

6) What is a strongly connected graph?

7) What is the algorithm to sync the nodes in a spanning tree? What’s the use of it?

8) What’s the time complexity of syncBFS?