

A Distributed Polylogarithmic Time Algorithm for Self-Stabilizing Skip Graphs

Christian Decker

Distributed Computing Seminar

Outline

1 Skip Graphs

2 ALG+

- Idea
- Rules

3 How it works

- Bottom-Up
- Top-Down
- Node join / departure

Overview

1 Skip Graphs

2 ALG+

- Idea
- Rules

3 How it works

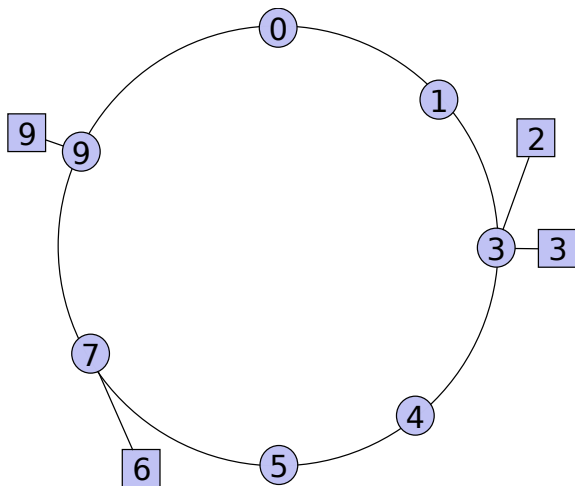
- Bottom-Up
- Top-Down
- Node join / departure

What is a DHT and what is it used for?

Datastructure for storage of data.

- Scalable
- Decentralized
- High fault tolerance

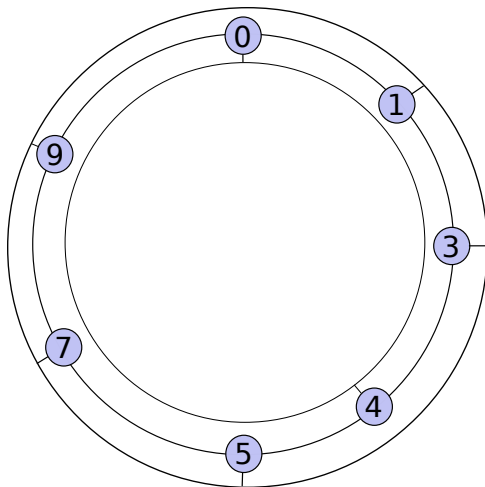
How to create a simple DHT



Weaknesses of our Ring design

- Long search/put/get operations
- Single failures result in unrecoverable state

Multiple Rings?



The idea behind Skip Graphs

We generalize the idea of having multiple lists:

- Each node participates in multiple rings/lists

The idea behind Skip Graphs

We generalize the idea of having multiple lists:

- Each node participates in multiple rings/lists
- Each node has a random String rs , of sufficient length

The idea behind Skip Graphs

We generalize the idea of having multiple lists:

- Each node participates in multiple rings/lists
- Each node has a random String rs , of sufficient length
- Each ring/list is identified by a prefix of length i

The idea behind Skip Graphs

We generalize the idea of having multiple lists:

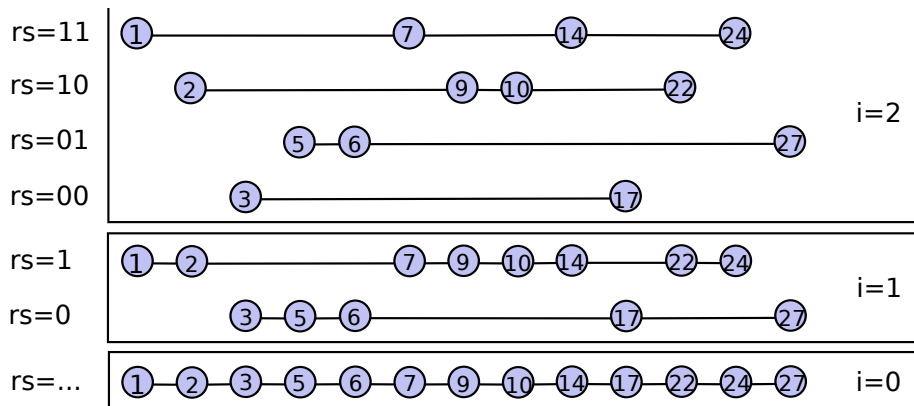
- Each node participates in multiple rings/lists
- Each node has a random String rs , of sufficient length
- Each ring/list is identified by a prefix of length i
- A node participates in a list if its rs matches the prefix of the list ($pre_i(u)$)

The idea behind Skip Graphs

We generalize the idea of having multiple lists:

- Each node participates in multiple rings/lists
- Each node has a random String rs , of sufficient length
- Each ring/list is identified by a prefix of length i
- A node participates in a list if its rs matches the prefix of the list ($pre_i(u)$)
- Each node has a successor ($succ_i(u)$) and predecessor ($pred_i(u)$) in each list it is participating in

The complete picture



Summary: Skip Graphs

- Logarithmic diameter
- Hypercubic-style Routing in $O(\log(n))$
- Not locally verifiable

Overview

1 Skip Graphs

2 ALG+

- Idea
- Rules

3 How it works

- Bottom-Up
- Top-Down
- Node join / departure

Goals

- Start from any weakly connected graph
- $O(\log^2(n))$ rounds to stabilize
- Fast node join and departure once stabilized

Idea

Divide the algorithm in two phases:

- 1 *Bottom-up* phase: create connected ρ -components for all prefixes ρ
- 2 *Top-Down* phase: sort each list by merging the the already sorted ρ_1 - and ρ_0 -components into a ρ -component

Idea

It's nothing more than distributed merge sort.

Tools

We have to slightly extend the original Skip Graph:

Extended Predecessor/Successor

Instead of considering just predecessor and successor from our prefix we already look ahead at the next bit:

$$\mathit{pred}_i^*(v, x) = \mathit{pred}(v, \{w \mid \mathit{pre}_{i+1}(w) = \mathit{pre}_i(v) \circ x\})$$

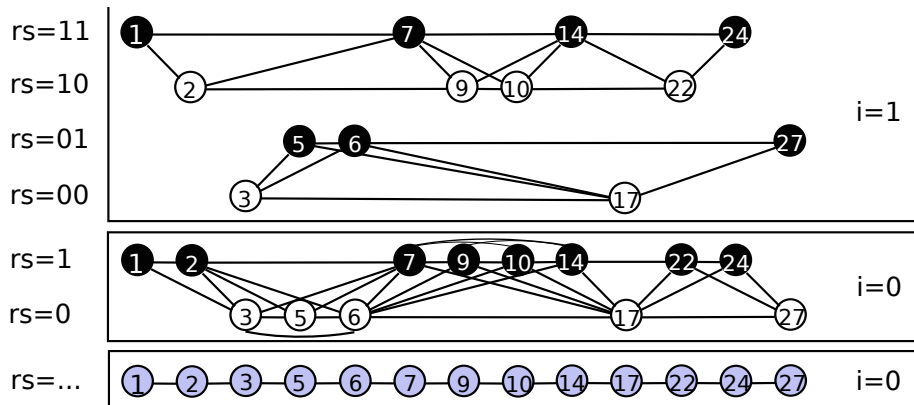
$$\mathit{succ}_i^*(v, x) = \mathit{succ}(v, \{w \mid \mathit{pre}_{i+1}(w) = \mathit{pre}_i(v) \circ x\})$$

Range

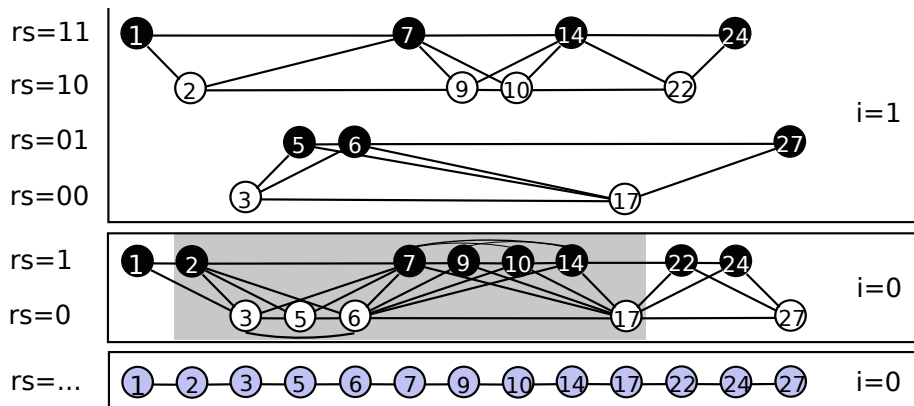
Take the farthest away of successor and predecessor. All nodes in between will be in the range and will be the neighbors of the current node:

$$v.\mathit{range}^*[i] = [\min\{\mathit{pred}_i^*(v, x)\}, \max\{\mathit{succ}_i^*(v, x)\}]$$

The final Picture



The final Picture



Tools

Stable / Temporary edges

Edges are either temporary or stable. Edges are considered stable if (from the local view of the node) it will appear in the finished Skip+ Graph. Stable edges are represented by a local flag $u.F(v) = 1$

ρ -component

A subgraph of all nodes sharing the prefix ρ .

ρ -Buddy

Each node has a Buddy at each level $|\rho| = i$ which differs at the last bit. These are used to pass temporary to better fitting candidates.

Basic operations

$insert(u, v)$ is the basic operation of the algorithm.
Any node w issues an $insert(u, v)$ operation to a node u telling it to add v to its neighborhood.

Rounds

- Preprocessing
 - ▶ Process all $insert(u, v)$ requests by adding them as temporary ($u.F(v) = 0$)
 - ▶ Check liveness of neighbors and remove failed ones
 - ▶ For every i determine $pred_i(u, 0)$, $pred_i(u, 1)$, $succ_i(u, 0)$ and $succ_i(u, 1)$
 - ▶ Send state updates to neighbors
- Execute Rules according to local state

Rules

Rule 1a: Create reverse edges

For every stable edge (u, v) , u sets $u.F(v) = 1$ and initiates an *insert* (v, u)

Rules

Rule 1a: Create reverse edges

For every stable edge (u, v) , u sets $u.F(v) = 1$ and initiates an $insert(v, u)$

Rule 1b and 1c: Introduce Stable Edges

u initiates $insert(v, w)$ and $insert(w, v)$ for every neighbors w in the range of v ($pre_i(v) = pre_i(w)$ and $w.id \in v.range[i]$)

Rules

Rule 2: Forward Temporary Edges

Every temporary edge (u, v) is forwarded to a stable neighbor of u that has the largest common prefix with $v.rs$

Rules

Rule 3a: Introduce All

Every node u that has changed its stable edge set (destabilizing or stabilizing edges) they introduce all neighbors with each other.

$$\text{insert}(u, v) \quad u, v \in N(w)$$

Rules

Rule 3a: Introduce All

Every node u that has changed its stable edge set (destabilizing or stabilizing edges) they introduce all neighbors with each other.

$$\text{insert}(u, v) \quad u, v \in N(w)$$

Rule 3b: Linearize

u identifies stable neighbors (v_1, \dots, v_k) , with common prefix of length i , orders them by $v_k.id$ and executes $\text{insert}(v_1, v_2)$, $\text{insert}(v_2, v_3)$, \dots , $\text{insert}(v_{k-1}, v_k)$

Overview

1 Skip Graphs

2 ALG+

- Idea
- Rules

3 How it works

- Bottom-Up
- Top-Down
- Node join / departure

How it works: Bottom-up

Remember that we want to create connected components for each non-trivial prefix ρ

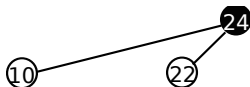
ρ -connected graphs stay connected

If a and b are ρ -connected at time t_0 , then they'll be ρ -connected at any $t > t_0$

How it works: Bottom-up

σ -V-Links

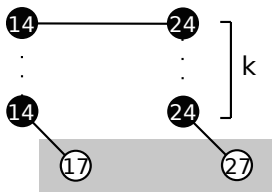
If we have two nodes u and v with prefix $\rho x = \sigma$ and a node w with prefix $\rho \bar{x}$ and $u, v \in N(w)$, then u and v are said to be σ -V-Linked. If two nodes are σ -V-Linked they'll be σ -connected in the next round.



How it works: Bottom-up

σ -k-Bridges

Two nodes a and b with prefix $\sigma = \rho x$ are in different components. c (d) is a stable neighbor of a (b) with prefix $\rho \bar{x}$. c and d are connected at level $|\rho| + k$.



How it works: Bottom-up

(σ, k) -pre-components

Two nodes in different $\rho X = \sigma$ components are in a (σ, k) -pre-component if G_ρ , each has a buddy in $\rho \bar{X}$ and they are connected (directly, via a σ -V-Link or via a (σ, k') -bridge with $k' \leq k$). After 4 rounds a and b are σ -connected.



How it works: Bottom-up

Bubbling up temporary edges

A temporary edge (u, v) at level l either stabilizes, or forwarded to $l + 1$.

How it works: Bottom-up

Connecting higher levels

If G_ρ is connected at time t then at time $t + (H - |\rho|) + O(\log(n))$ the graphs $G_{\rho 0}$ and $G_{\rho 1}$ will also be connected.

How it works: Bottom-up

Connecting higher levels

If G_ρ is connected at time t then at time $t + (H - |\rho|) + O(\log(n))$ the graphs $G_{\rho 0}$ and $G_{\rho 1}$ will also be connected.

Connecting every level

Since every node participates in $O(\log(n))$ levels, connecting each level takes $O(\log^2(n))$

How it works: Top-down

We merge already sorted to build the lower levels:

How it works: Top-down

We merge already sorted to build the lower levels:

i-finished

The graph is *i*-finished if $\forall \rho$ with $|\rho| = i$, G_ρ contains all edges of the final Skip+ Graph and is stably connected to all nodes in his range.

How it works: Top-down

We merge already sorted to build the lower levels:

i-finished

The graph is *i*-finished if $\forall \rho$ with $|\rho| = i$, G_ρ contains all edges of the final Skip+ Graph and is stably connected to all nodes in his range.

Moving down

If at time t the graph is *i*-finished, at time $t + 3$ it will be $(i - 1)$ -finished.

Summary

- Connecting all levels takes $O(\log^2(n))$ rounds
- Ordering all levels adds $O(\log(n))$ rounds
- Overall complexity is $O(\log^2(n))$

Node join / departure

- The Skip+ Graph has degree $O(\log(n))$
- When a node joins/leaves only the neighbors are involved
- At most $O(\log^4(n))$ work (edge inserts)
- Joins / Leaves can be handled $O(\log(n))$ rounds

References

- *Skip Graphs*, James Aspnes and Gauri Shah, *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384-393, Baltimore, MD, USA, 12-14 January 2003
- *A distributed polylogarithmic time algorithm for self-stabilizing skip graphs*, Riko Jacob, Andrea Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig, *PODC 09: Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 131-140, New York, NY, USA, 2009. ACM

Questions?