Advanced Topics in DS:

Distributed Storage

Presented by

Tiziano, Stephane, Carlos, Yaseen, Yuna, Emir

Recent years have witnessed an incredible amount of user-generated content through applications such as social networks, blogs and multimedia-sharing services.

The key challenge that research community as well as Industry is facing is to design a cost efficient storage system to cope with this data explosion.

A Distributed Storage System formed, by networking together a large number of, inexpensive and unreliable, storage devices provides one such alternative to store such a massive amount of data with high reliability and availability.

Introduction

What is Distributed Data Storage?

It's a **computer network** in which information is stored in **more than one node**, often in a **replicated** fashion.

How can we achieve this subject?

- Consistent Hashing
- Hypercubic Networks
- DHT(Distributed Hash Table) & Churn

Consistent Hashing

Consistent Hashing

- What is Consistent Hashing?

  It's a special kind of hashing such that when a hash table is resized, only **K/n** keys need to be remapped on average, where **K** is number of keys and **n** the number of slots

- How can we realize this hash table?

  Starts off as a **sparse ring** data structure.

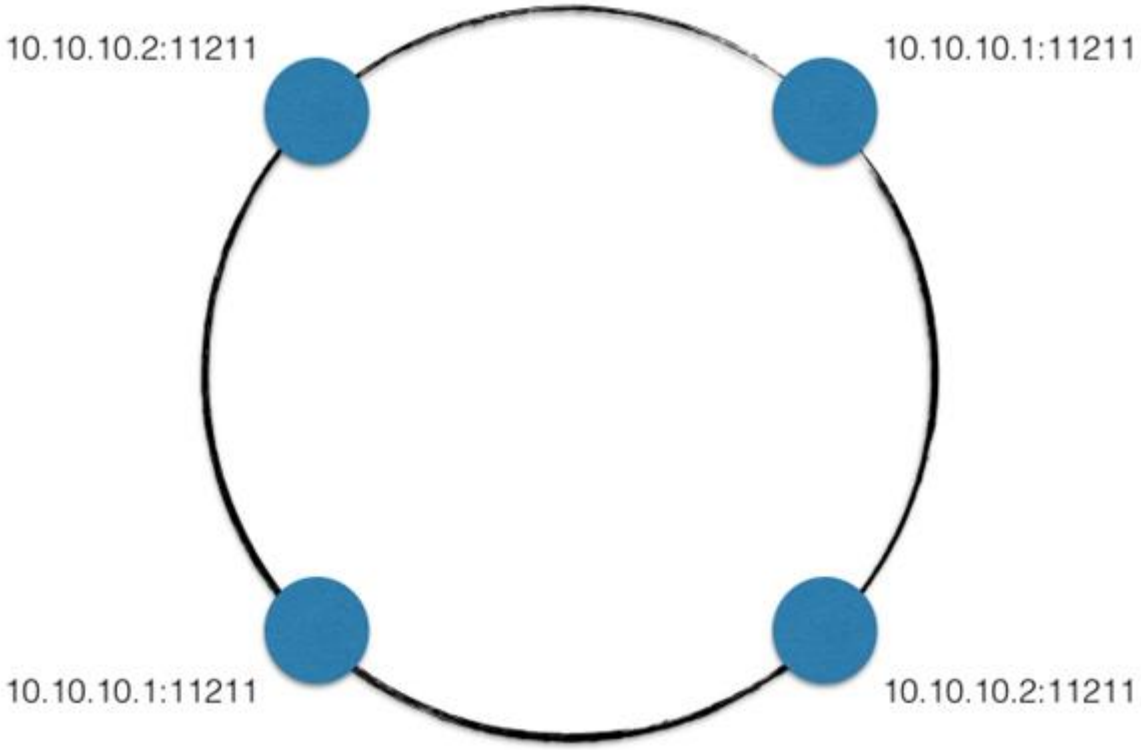
  A **SortedMap[K, V]** where K is the offset in the ring and V the value at that position.

  Every node has **N entries** in the ring.

  N is commonly referred as the node's **weight** and correspond to the probability of the node to be selected.

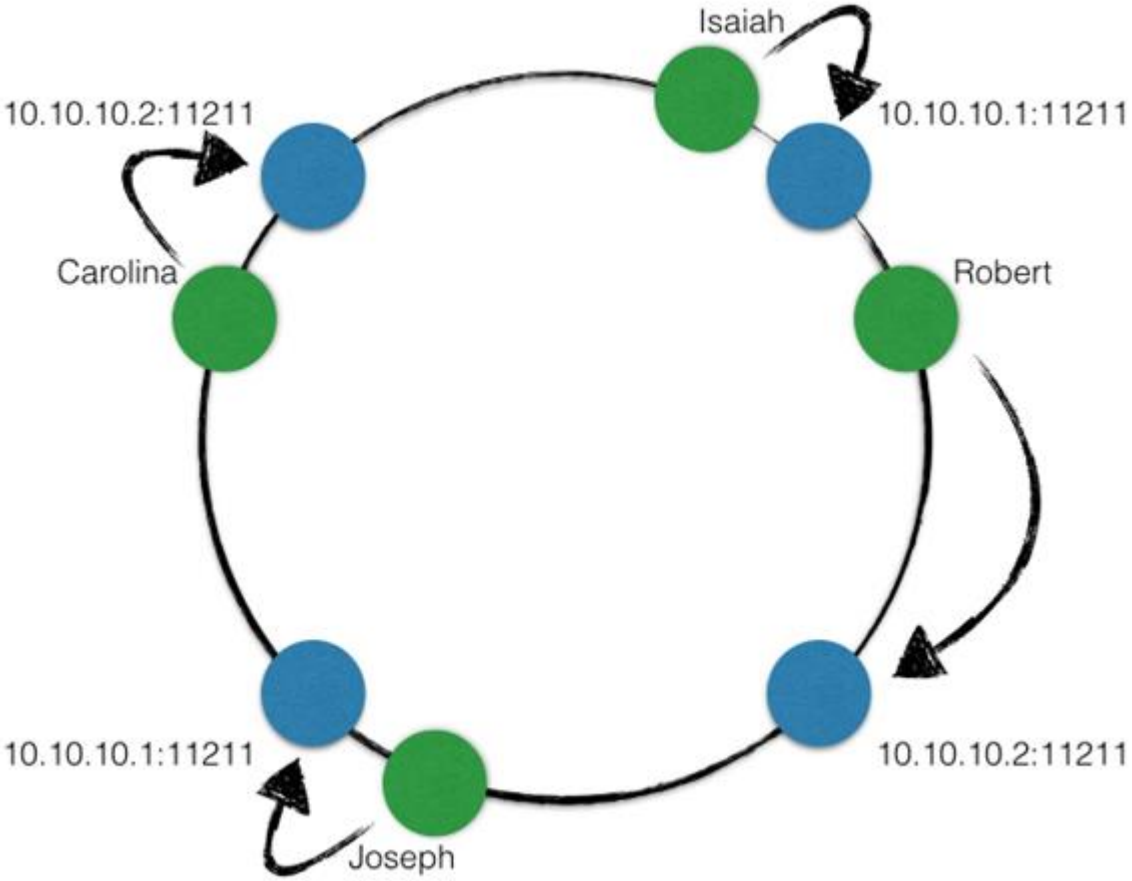
  Every node has a probability **P = W/S** to be selected, where W is its weight and S the Sum of all nodes' entries.

Example



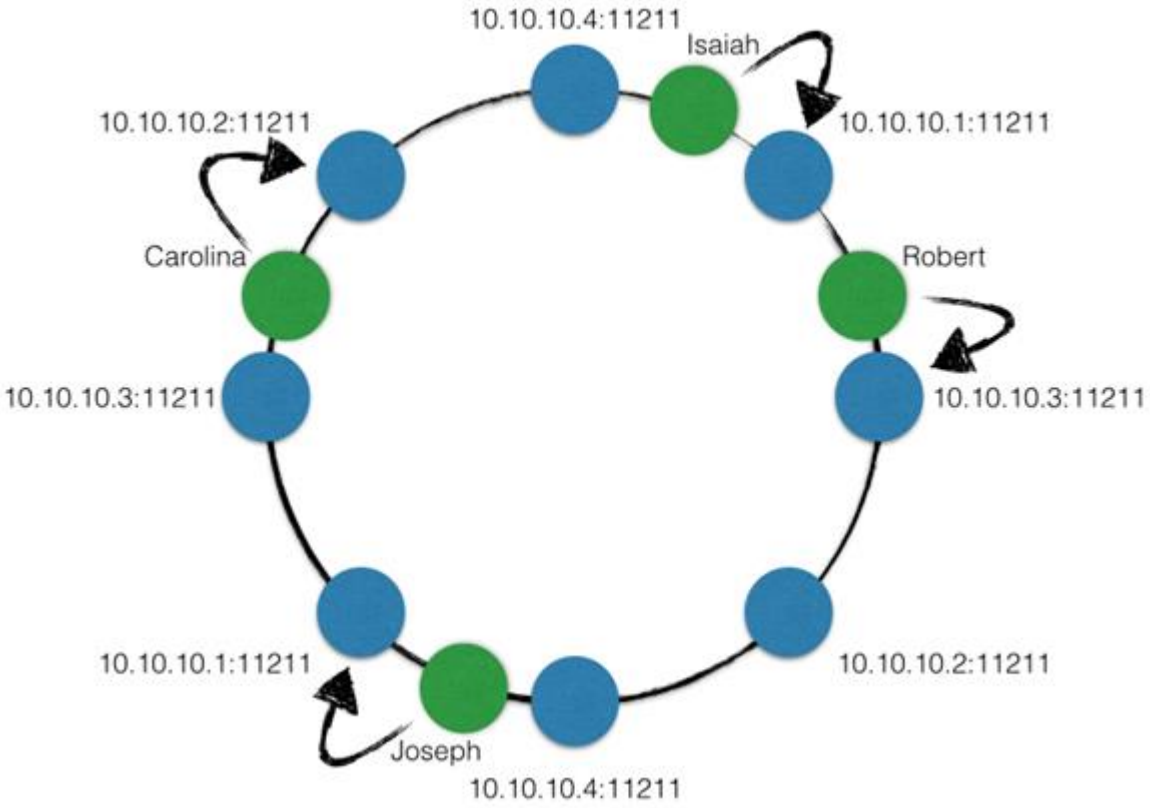
Let's start with a simple hash ring of **2 nodes**, each with a **weight** equals to **2**.
Every node is put in the Map applying a **known set of hash functions** **(h_1, …, h_k)** to identify its point around the circle.

Example (cont.)



Now, to map a key to the corresponding node, we apply the **same set of hash functions** used to map the nodes.
Every key-point is **associated** with the first **clockwise node-point** found. (as shown in the picture)

Example (cont.)



- Now, **adding 2 nodes** with the **same weight** as previous, that's what happens.
- As we can see, only the «Robert» key has to be remapped.
- A similar situation takes place in case some node become unavailable and all its entries have to be removed

In Summary

- Consistent hashing is based on mapping each object to a point on the **edge of a circle**
- The system maps each available machine to **many pseudo-randomly distributed points** on the edge of the same circle
- the system **finds** the location of that object's **key**; then **walks around** the circle until falling into the first bucket
- If a bucket becomes unavailable then the points it maps to will be removed; Requests now map to the next highest points
- The portion of the keys associated with each bucket can be altered by altering the number of angles (**weight** of the node) that bucket maps to

Theorem 21.2 (Consistent Hashing).
In expectation, the Algorithm stores each object **kn/m** times.
Proof: While it is possible that some object does not hash closest to a node for any of its hash functions, this is highly unlikely: For each node (n) and each hash function (k), each object (m) has about the same probability (1/m) to be stored. By linearity of expectation, a movie is stored kn/m times, in expectation.

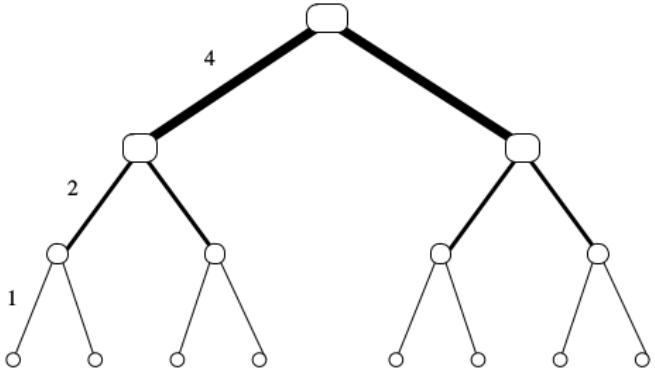
Hypercubic Networks

Topology Properties

- **Homogeneous**
  - No dominant node
  - No single point of failure
- **Non-anonymous**
  - ID $\in [0,1[$
  - Hash functions
- **Small node degree**
  - Polylogarithmic in n: $O(log^k(n))$
  - Deal with churn
- **Small network diameter**
  - Polylogarithmic in n
  - Easy routing

Fat Trees

- **Trees:**
  - Easy routing: single path between any pair of nodes
  - Bottleneck: non-homogeneous (e.g. the root)
- **Fat-trees:**
  - each edge *(u,v)* has a capacity proportional to the number of leaves in the subtree rooted at *v*



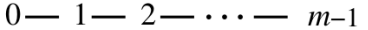
Mesh

- **Definition** of the (m,d)-mesh:
  - $m, d \in \mathbb{N}$
  - Graph $G = (V, E)$ with

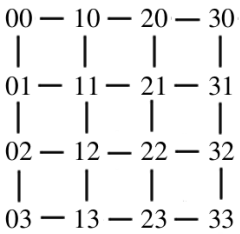
$$V = [m]^d = \{0, \ldots, m-1\}^d$$

$$E = \left\{ \{(a_1, \ldots, a_d), (b_1, \ldots, b_d)\} \mid a_i, b_i \in [m], \sum_{i=1}^{d} |a_i - b_i| = 1 \right\}$$

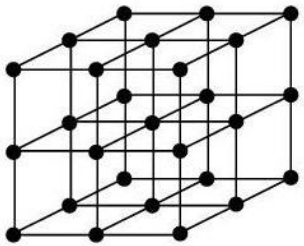
- **Examples:**



$M(m,1)$

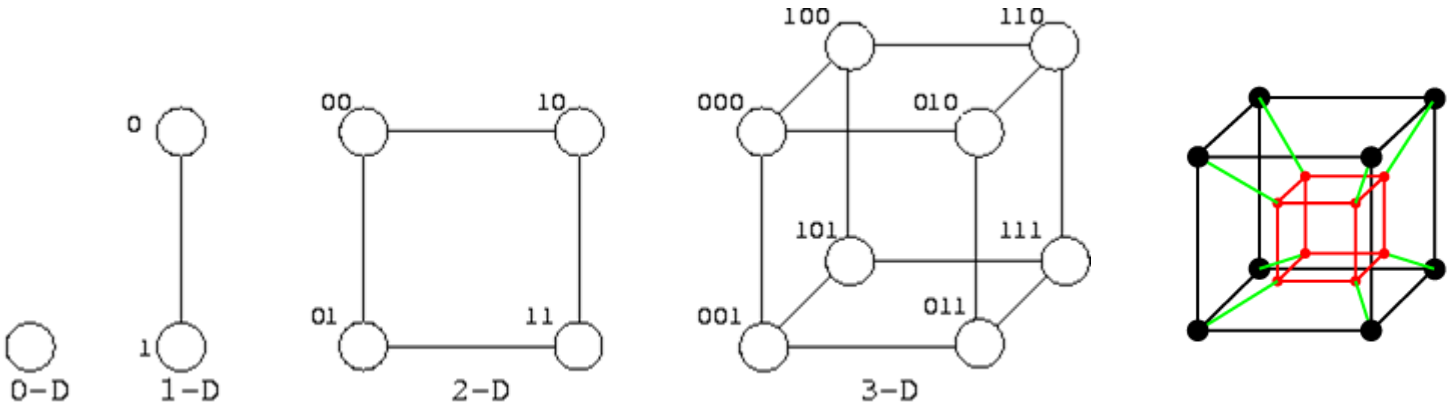


$M(4,2)$



Hypercubes

- **Definition:** a d-dimensional hypercube is a (2,d)-mesh
- **Properties:**
  - dimension *d*
  - 2^d nodes
  - node degree *d*
  - network diameter *d*
  - routing: fix wrong bit, one at the time (multiple paths)
- **Examples:**

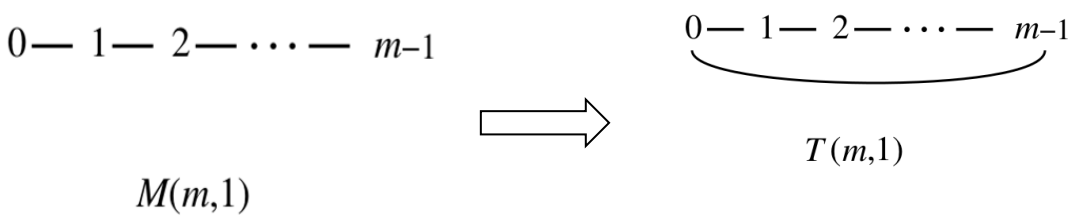


Torus

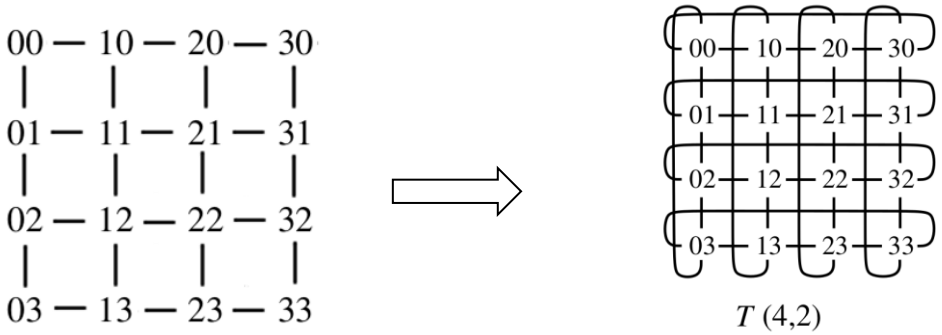
- **Definition:** a (m,d)-torus is an (m,d)-mesh with additional *wrap-around edges* between nodes (for all *i*)

$(a_1, \ldots, a_{i-1}, m-1, a_{i+1}, \ldots, a_d)$ $(a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_d)$ and

- **Examples:**



$M(m,1)$ $T(m,1)$



$M(4,2)$ $T(4,2)$

NB: since *m-1=1*,

H(d) = M(2,d)

= T(2,d)

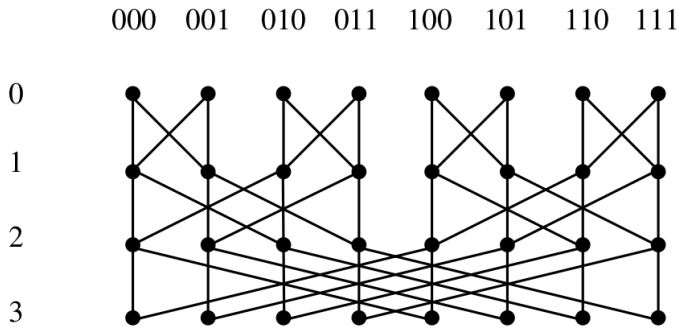
Butterfly

- **Definition:** the d-dimensional butterfly *BF(d)* is a graph *G=(V, $E_1 \cup E_2$)* with $V = [d+1] \times [2]^d$

$$E_1 = \{\{(i,\alpha),(i+1,\alpha)\} \mid i \in [d],\ \alpha \in [2^d]\}$$

$$E_2 = \{\{(i,\alpha),(i+1,\beta)\} \mid i \in [d],\ \alpha,\beta \in [2^d], |\alpha - \beta| = 2^i\}$$

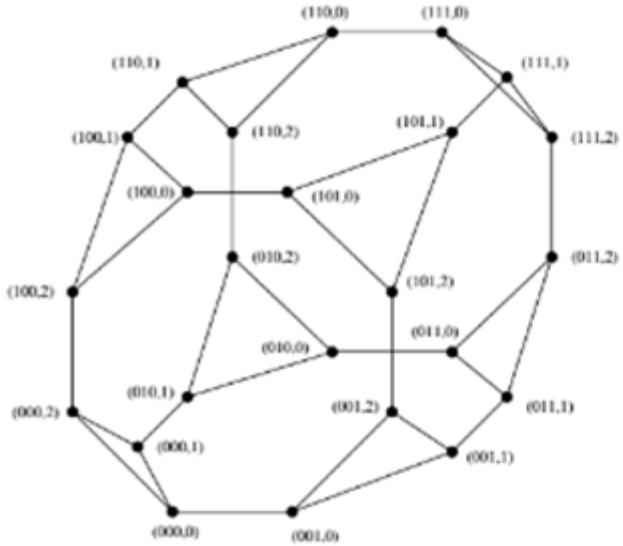
- Level *i*: $\{(i,\alpha) \mid \alpha \in [2]^d\}$
- Wrap-around butterfly: $(d,\alpha) = (0,\alpha)\ for\ all\ \alpha \in [2]^d$
- Property: constant node degree
- **Examples**:



Definition 21.9 Cube-Connected-Cycles

Let d ∈ N. The cube-connected-cycles network CCC(d) is a graph with node set $V = \{(a,p) \mid a \in [2]^d, p \in [d]\}$ and edge set

$$E = \{\{(a,p),(a,(p+1) mod\ d)\} | a \in [2]^d, p \in [d]\}$$
$$\cup \{\{(a,p),(b,p)\} | a,b \in [2]^d, p \in [d], a = b\ except\ for\ a_p\}$$



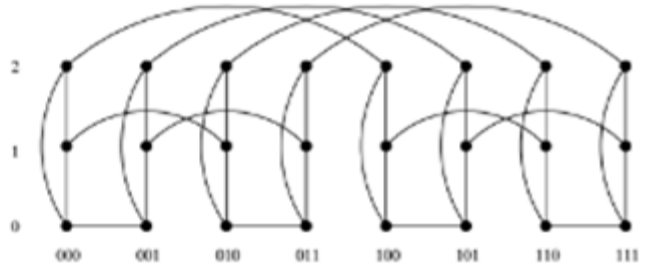


Figure 21.10: The structure of CCC(3).

Remarks:

- Two possible representations of a CCC can be found in Figure 21.10.
- The shuffle-exchange is yet another way of transforming the hypercubic interconnection structure into a constant degree network.

Definition 21.11 Shuffle-Exchange

Let d ∈ N. The d-dimensional shuffle-exchange SE(d) is defined as an undirected graph with node set $V = [2]^d$ and an edge set $E = E_1 \cup E_2$ with

$$E_1 = \{\{(a_1,\ldots,a_d),(a_1,\ldots,\bar{a}_d)\} \mid (a_1,\ldots,a_2) \in [2]^d,\ \bar{a}_d = 1 - a_d\}$$

and

$$E_2 = \{\{(a_1,\ldots,a_d),(a_d,a_1,\ldots,a_{d-1})\} / (a_1,\ldots,a_d) \mid (a_1,\ldots,a_d) \in [2]^d\}$$

Figure 21.12 shows the 3 and 4 dimensional shuffle-exchange graph.

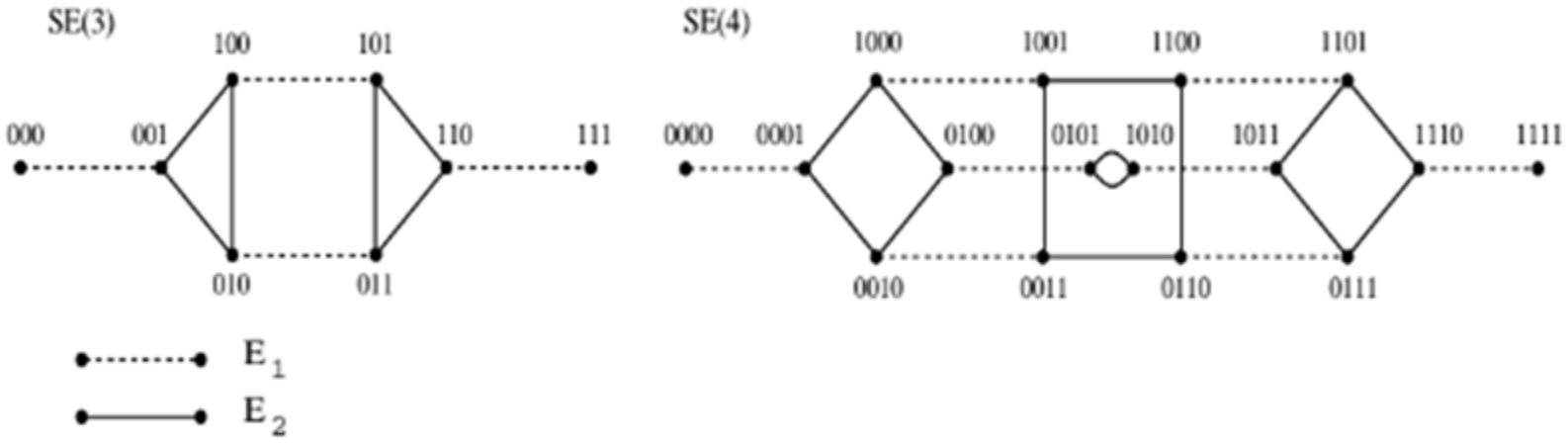


Figure 21.12: The structure of SE(3) and SE(4).

Definition 21.13 DeBruijn

The b-ary DeBruijn graph of dimension d DB(b, d) is an undirected graph G=(V,E) with node set $V = \{v \in [b]^d\}$ and edge set E that contains all edges {v, w} with the property that $w \in \{(x, v_1, \ldots, v_{d-1}) : x \in [b]\}$, where $v = \{v_1, \ldots, v_d\}$.

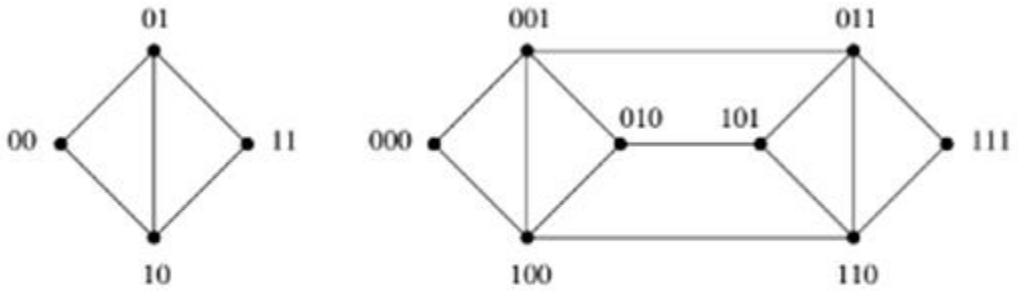


Figure 21.14: The structure of $DB(2,2)$ and $DB(2,3)$.

Remarks:

- Two examples of a DeBruijn graph can be found in Figure 21.14.
- There are some data structures which also qualify as hypercubic networks. An example of a hypercubic network is the skip list, the balanced binary search tree for the lazy programmer.

Skip List

- An ordinary ordered linked list of objects, augmented with additional forward links.
- Ordinary linked list is level 0, and in addition, every object is promoted to level 1 with probability 1/2.
- As for level 0, all level 1 objects are connected by a linked list.
- In general, every object on level i is promoted to the next level with probability 1/2.
- A special start-object points to the smallest/first object on each level.

Remarks:

- Search, insert, and delete can be implemented in O(log n) expected time in a skip list.
- The randomization can easily be discarded, by deterministically promoting a constant fraction of objects of level i to level i+1,for all i.
- There are obvious variants of the skip list, e.g., the skip graph, balanced binary tree.
- More generally, how are degree and diameter of Definition 21.3 related? See in theorem

Theorem: Every graph of maximum degree d > 2 and size n must have a diameter of at least ⌈(log n)/(log(d-1))⌉ -2.

Proof. Suppose we have a graph G = (V;E) of maximum degree d and size n. Start from any node v ∈ V . In a first step at most d other nodes can be reached. In two steps at most d (d-1) additional nodes can be reached. Thus, in general, in at most k steps at most

$$1 + \sum_{i=0}^{k-1} d \cdot (d-1)^i = 1 + d \cdot \frac{(d-1)^k - 1}{(d-1) - 1} \leq \frac{d \cdot (d-1)^k}{d-2}$$

ensure that v can reach all other nodes in V within k steps. Hence,

$$(d-1)^k \geq \frac{(d-2) \cdot n}{d} \quad \Leftrightarrow \quad k \geq \log_{d-1}((d-2) \cdot n/d)$$

Since $\log_{d-1}((d-2)/d) > -2$ for all d > 2, this is true only if k >= ⌈(log n)/(log(d-1))⌉-2.

Remarks

- In other words, constant-degree hypercubic networks feature an asymptotically optimal diameter.
- Other hypercubic graphs manage to have a different trade-offs between node degree and diameter.The pancake graph, for instance, minimizes the maximum of these with d = k =θ(log n/log log n). The ID of a node u in the pancake graph of dimension d is an arbitrary permutation of the numbers 1,2, . . . ,d. Two nodes u, v are connected by an edge if one can get the ID of node v by taking the ID of node u, and reversing (flipping) the first i numbers of u's ID. For example, in dimension d = 4, nodes u = 2314 and v = 1324 are neighbors.
- There are a few other interesting graph classes which are not hypercubic networks, but nevertheless seem to relate to the properties of Definition 21.3.Small-world graphs.
- Expander graphs (an expander graph is a sparse graph which has good connectivity properties, that is, from every not too large subset of nodes you are connected to an even larger set of nodes) are homogeneous, have a low degree and small diameter. However, expanders are often not routable.

DHT & Churn

Definition - DHT (Distributed Hash Table)

A distributed hash table (DHT) is a distributed data structure that implements a distributed storage.

A typical DHT consists of a search for a key,

an insert (key, object) operation

And sometimes, a delete (key) operation.

Notable distributed networks that use DHTs include BitTorrent's distributed tracker.

the Internet domain name system (DNS) is essentially a DHT.

Emphasized properties in DHT

- Autonomy and decentralization: the nodes collectively form the system without any central coordination.
- Fault tolerance: the system should be reliable even with nodes continuously joining, leaving and failing.
- Scalability: the system should function efficiently even with thousands or millions of nodes.

A key technique used to achieve these goals is that any one node needs to coordinate with only a few other nodes in the system – most commonly, $O(\log n)$ of the n participants – so that only a limited amount of work needs to be done for each change in membership.

Hypercubes and DHT

A DHT can be implemented as a hypercubic overlay network with nodes having identifiers such that they span the ID space [0,1).

A hypercube can directly be used for a DHT. Just use a globally known set of hash functions hi, mapping movies to bit strings with d bits.

Other hypercubic structures may be a bit more intricate when using it as a DHT: The butterfly network, for instance, may directly use the d+1 layers for replication, i.e., all the d+1 nodes are responsible for the same ID.

Other hypercubic networks, e.g. the pancake graph, might need a bit of twisting to find appropriate IDs.

Bootstrap Problem

We assume that a joining node knows a node which already belongs to the system. This is known as the bootstrap problem.

Typical solutions are: If a node has been connected with the DHT previously, just try some of the previous nodes. Or the node may ask some authority for a list of IP addresses (and ports) of nodes that are regularly part of the DHT.

Analyzation

When the sketched DHT is analyzed against an adversary that can crash a fraction of random nodes:

- We assume that joins and leaves occur in a worst-case manner. We think of an adversary that can remove and add a bounded number of nodes (at most add and/or remove O(logn) nodes); the adversary can choose which nodes to crash and how nodes join.
- The adversary does wait until the system is recovered before it crashes again. The system is never fully repaired but always fully functional. In particular, the system is resilient against an adversary that continuously attacks the "weakest part" of the system.

This model covers an adversary remaining or newly joining nodes towards the areas under attack. Which repeatedly takes down nodes by a distributed denial of service attack, however only a logarithmic number of nodes at each point in time. The algorithm relies on messages being delivered timely, in at most constant time between any pair of operational nodes, i.e., the synchronous model. Using the trivial synchronizer this is not a problem. We only need bounded message delays in order to have a notion of time which is needed for the adversarial model. The duration of a round is then proportional to the propagation delay of the slowest message.

DHT Algorithm characteristics

Conditions

- Set of hashing functions h_i
- Hypercube

Settings

- Each hypercube virtual node consisting of Θ log(n) nodes
- Nodes connected to all other nodes of their hypernode, and to nodes of their neighboring hypernodes

Possible

- Churn can cause that some of nodes might need to change to another hypernode s.t. all hypernodes own the same number of nodes at all times

Hypercube dimension

- In case the number of nodes increases or decreases over a given pre-established limit - the dimension of the hypercube increases or decreases by one respectively

DHT Algorithm - remarks

Logarithmic number of hypercube neighbors means that each node has

$\Theta(\log^2 n)$

Balancing of nodes among the hypernodes can be seen as a dynamic token distribution problem on the hypercube

Goal

Distribute the tokens along the edges of the graph

- striving for equal distribution of tokens

When tokens are moved around, an adversary constantly inserts and deletes tokens

DHT Algorithm - remarks

Two basic components of the storage system:

- an algorithm which performs the dynamic token distribution
- an information aggregation algorithm used to estimate the number of nodes in the system / to adapt the dimension of the hypercube accordingly

DHT with Churn

A fully scalable, efficient distributed storage system which tolerates O(log n) worst-case joins and/or crashes per constant time interval.

As in other storage systems, nodes have O(log n) overlay neighbors, and the usual operations (e.g., search, insert) take time O(log n).

Concluding remarks

Storing data has evolved during the years in order to accommodate the raising needs of companies and individuals. We are now reaching a tipping point at which the traditional approach to storage – the use of a stand-alone, specialized storage box – no longer works, for both technical and economical reasons.

At present the best approach to satisfying current demands for storing data seems to be distributed storage.

In our presentation we demonstrated different ways of approaching distributed storage.