

Load Balancing in Structured Overlay Networks



Tallat M. Shafaat
tallat(@)kth.se

Overview

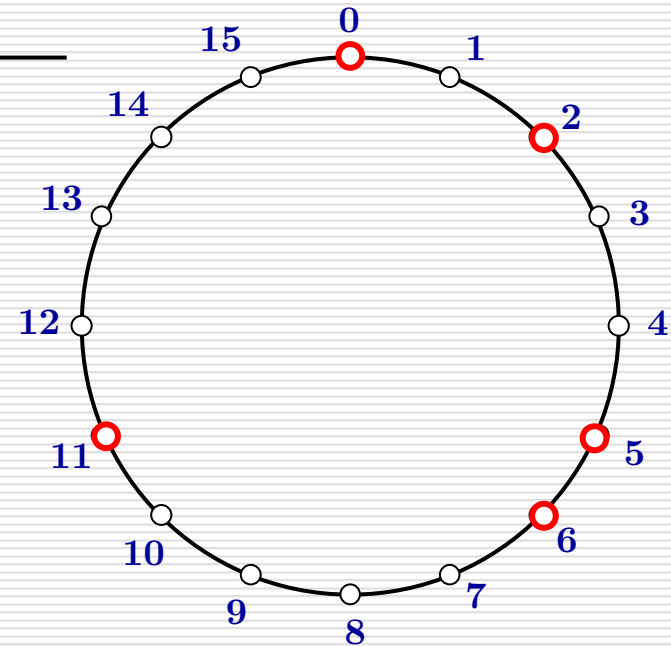
- Background
- The problem : load imbalance
- Causes of load imbalance
- Solutions

But first, some slides from previous lectures

How to construct a DHT (Chord)?

- Use a logical name space, called the *identifier space*, consisting of identifiers $\{0,1,2,\dots, N-1\}$
- Identifier space is a logical ring modulo N
- Every node picks a random identifier through Hash H

- Example:
 - Space $N=16 \{0,\dots,15\}$
 - Five nodes a, b, c, d
 - a picks 6
 - b picks 11
 - c picks 0
 - d picks 5
 - e picks 2



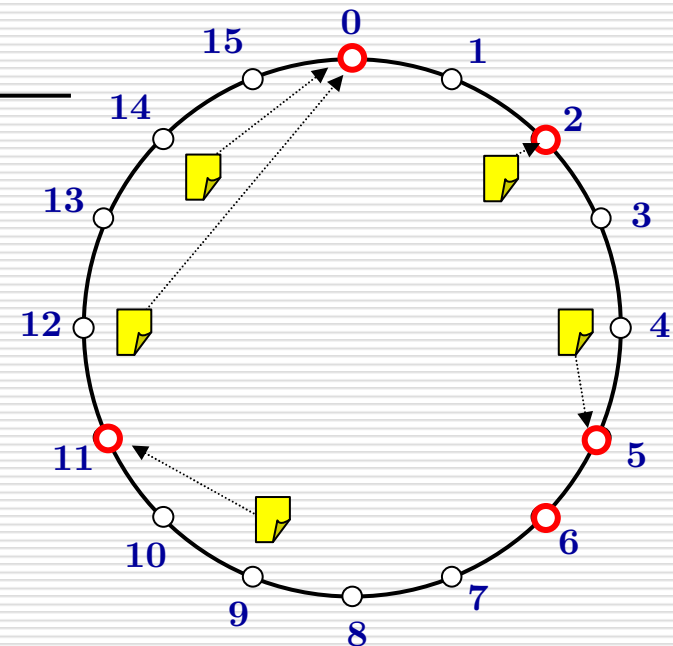
Where to store data (Chord) ?

- Use globally known **hash function**, H
- Each item $\langle \text{key}, \text{value} \rangle$ gets identifier = $H(\text{key})$
- Store each item at its successor
 - Node n is **responsible** for item k

Key	Value
Alexander	Berlin
Marina	Gothenburg
Peter	Louvain la neuve
Seif	Stockholm
Ali	Stockholm

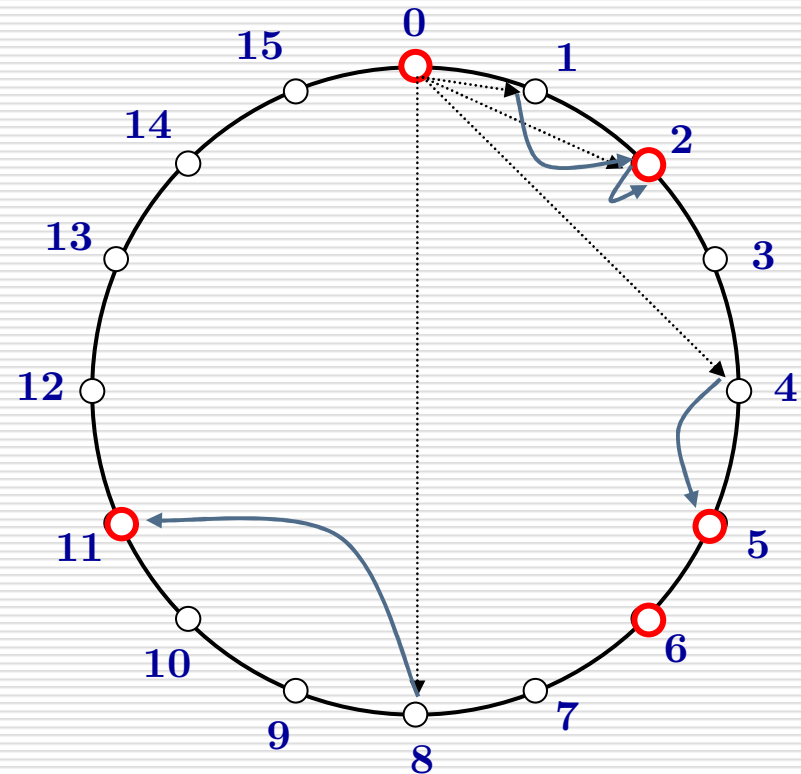
□ Example

- $H(\text{"Alexander"})=4$
- $H(\text{"Marina"})=12$
- $H(\text{"Peter"})=2$
- $H(\text{"Seif"})=9$
- $H(\text{"Ali"})=14$



Speeding up lookups (Chord)

- If only pointer to $\text{succ}(n+1)$ is used
 - Worst case lookup time is N , for N nodes
- Improving lookup time (finger/routing table)
 - Point to $\text{succ}(n+1)$
 - Point to $\text{succ}(n+2)$
 - Point to $\text{succ}(n+4)$
 - Point to $\text{succ}(n+8)$
 - ...
 - Point to $\text{succ}(n+2^{M-1})$
- Distance always halved to the destination
- Routing entries = $\log_2(N)$
- $\log_2(N)$ hops from any node to any other node



Summing up

□ Consistent hashing

- Identifier space N
- Nodes take an identifier randomly from N
- Data items are stored under identifiers from N

□ Chord

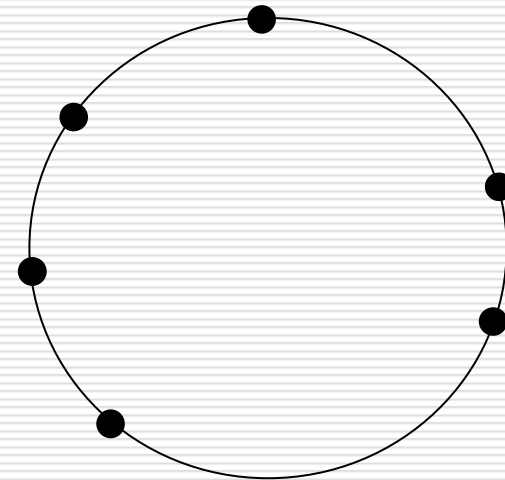
- Each node maintains $\log(n)$ distinct links to other nodes, where n is the number of nodes
- Lookups take $\log(n)$ hops

Definition of Load

- Load of a data item may refer to:
 - Number of bits required to store the item
 - Popularity of the item
 - Amount of processor time needed to serve the item
 - ...

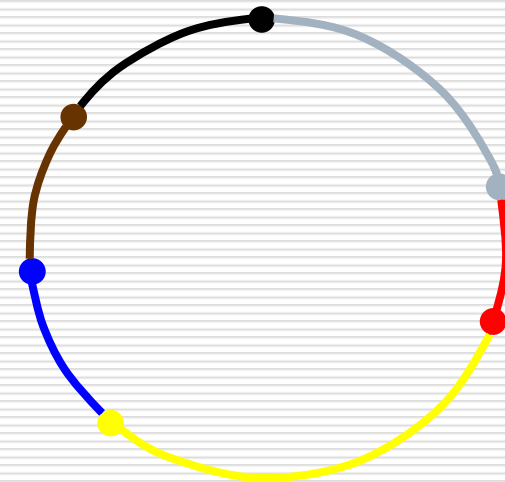
The problem: Load imbalance

- Node identifiers may not be balanced



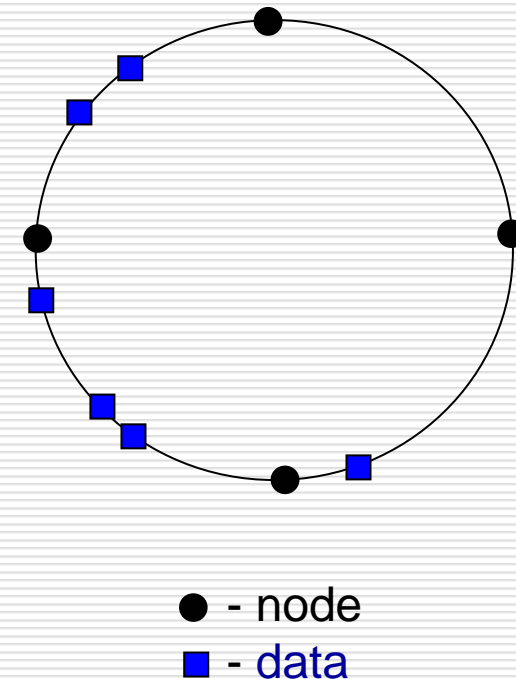
The problem: Load imbalance

- Node identifiers may not be balanced



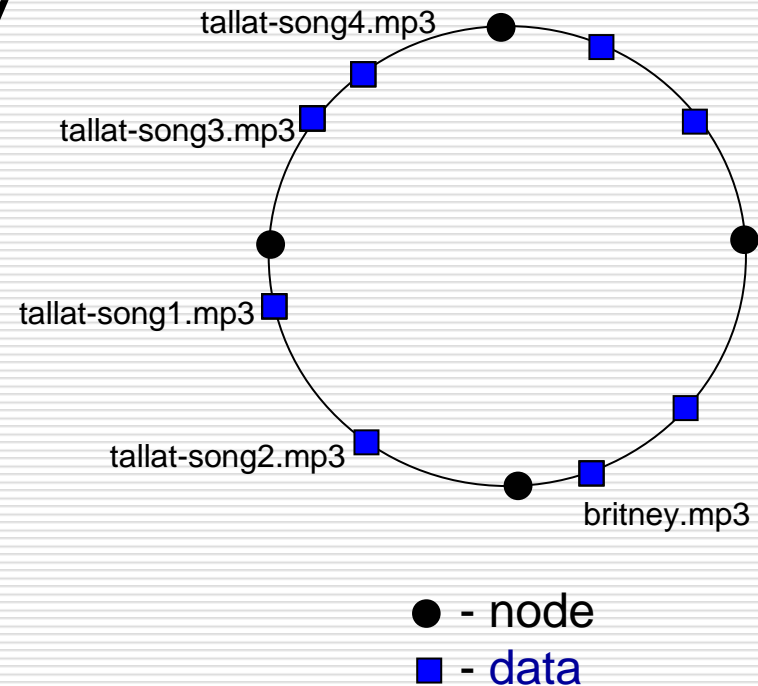
The problem: Load imbalance

- ❑ Node identifiers may not be balanced - *assume solved*
- ❑ Data identifiers may not be balanced



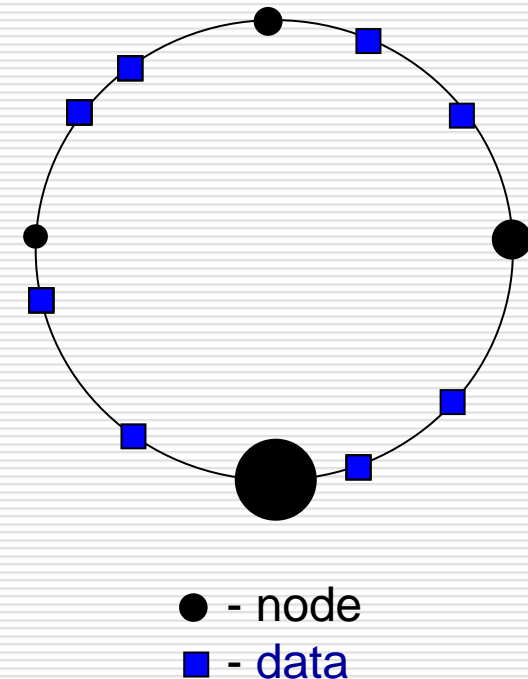
The problem: Load imbalance

- ❑ Node identifiers may not be balanced - *assume solved*
- ❑ Data identifiers may not be balanced - *assume solved*
- ❑ Hot spots



The problem: Load imbalance

- ❑ Node identifiers may not be balanced - *assume solved*
- ❑ Data identifiers may not be balanced - *assume solved*
- ❑ Hot spots - *assume solved*
- ❑ Heterogeneous nodes



Overview

- ☑ Background
- ☑ The problem : load imbalance
- ☑ Causes of load imbalance
- ☐ Solutions
 - Method 1: Virtual servers
 - Method 2: Without virtual servers

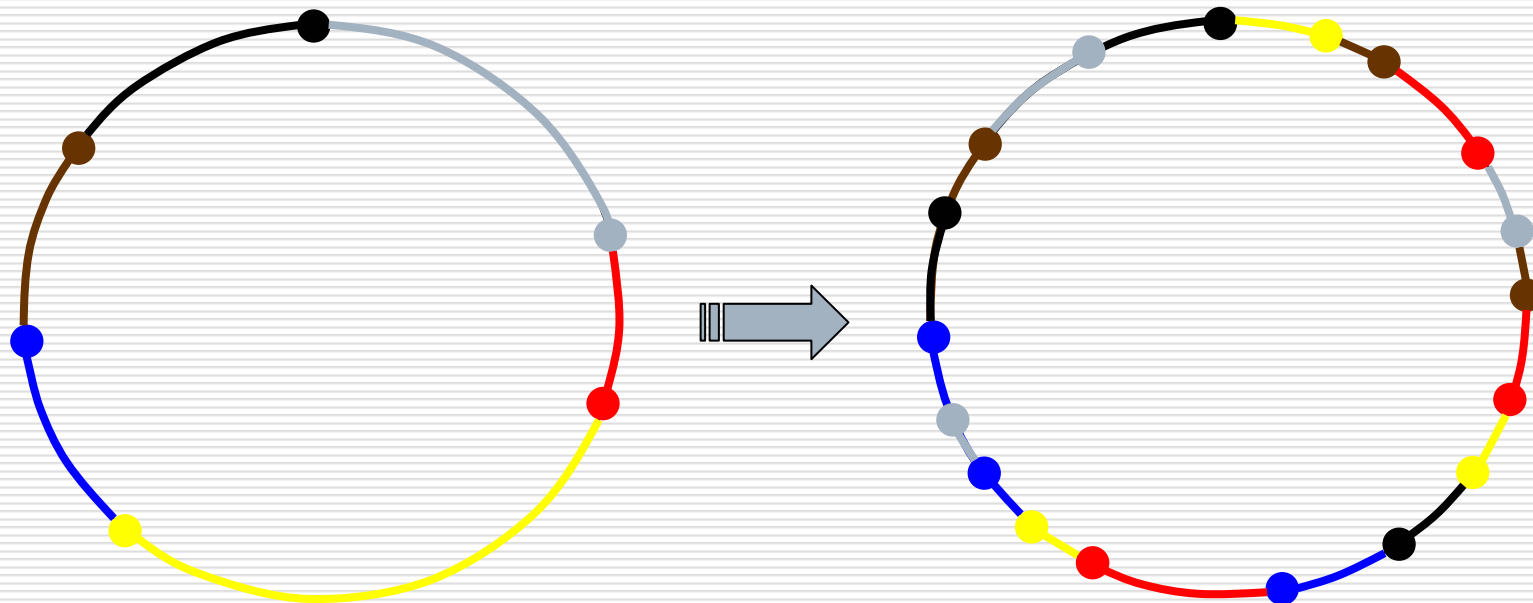
Assume uniform load distribution i.e. load of a node is proportional to size of id space it is responsible for

Load imbalance

- It has been shown that
 - Given
 - a homogenous system
 - with n nodes
 - If node and item identifiers are randomly chosen, then there is an $O(\log n)$ imbalance factor

Virtual Servers (VSs)

- Goal: **# of keys/node** should be uniform
- Each physical node picks multiple random identifiers
 - Each identifier represents a virtual server
 - Each node runs multiple virtual servers
- Each node responsible for noncontiguous regions



Virtual Servers (VSs)

- Basic idea for load balancing:
 - Move virtual servers from **heavily loaded** physical nodes to **lightly loaded** physical nodes
- Challenges:
 - Minimize the load imbalance
 - Minimize the amount of load moved
 - spoiler warning – spoiler warning
- Steps
 - Knowing if a node is overloaded or not
 - Generating a mapping of transferring VSs
 - Transferring VSs

VSs – Model

- n nodes
- l_i – load at n_i at a particular time
- c_i – capacity of n_i e.g. bandwidth, disk space

- μ_i – utilization of n_i

$$\mu_i = \frac{l_i}{c_i}$$

- when $\mu_i > 1$, n is overloaded, otherwise n is underloaded

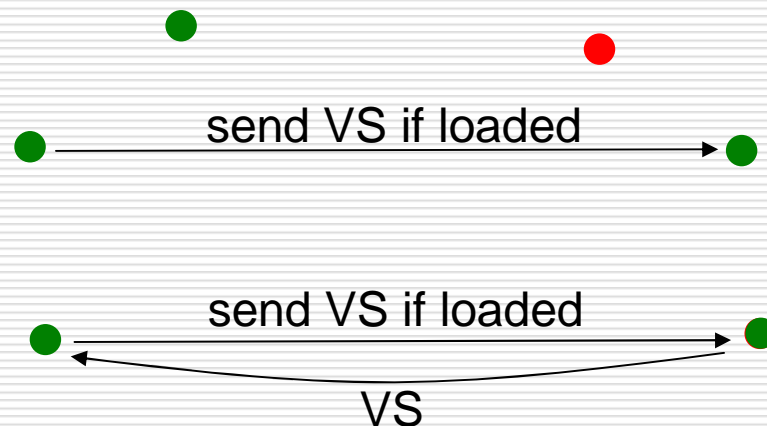
- μ – system utilization

$$\mu = \frac{\sum_{nodes\ n} l_n}{\sum_{nodes\ n} c_n}$$

VSs – Static load balancing schemes

□ One-to-one

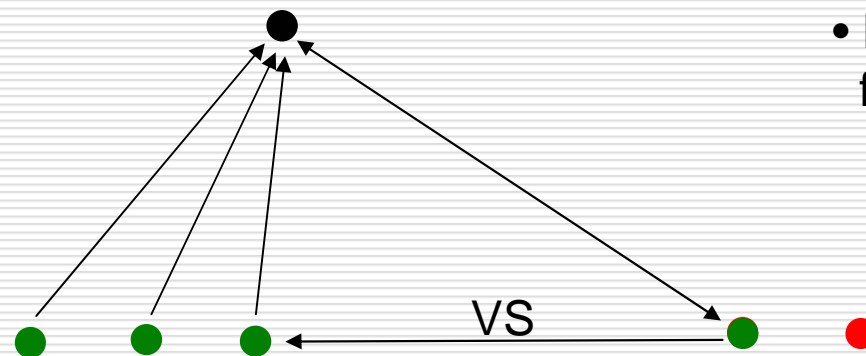
- Each lightly loaded node l periodically contacts a random node r
- If r is heavily loaded, virtual servers are transferred from r to l such that r becomes light without making l heavy



VSs – Static load balancing schemes

□ One-to-many

- Directory ***d*** has a random set of **light** nodes
- Each heavy node ***h*** gets some nodes from ***d***
- Some of ***h***'s virtual servers are then moved to one or more of the lighter nodes registered in the directory

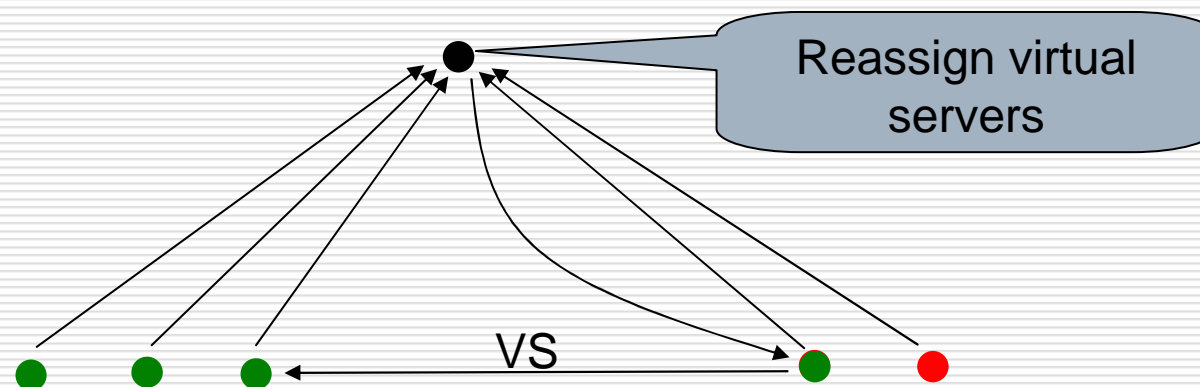


- Multiple directories for fault-tolerance

VSs – Static load balancing schemes

□ Many-to-many

- Each directory maintains load information for a set of both light and heavy nodes.
 - Each directory runs an algorithm to decide the reassignment of virtual servers from heavy nodes to light nodes
-



VSs – Hybrid scheme

- Combine
 - one-to-many AND many-to-many
- Each node:
 - periodically report loads and capacity to random directory
 - if directory suggests transfer of VSs, do it
- Each directory
 - store reports from nodes
 - periodically reassign VSs such that
 - load transfer should be minimum
 - do not overload another node
- Emergency action at node n
 - if n becomes overloaded, ask the directory for immediate reassignment of VSs

Virtual Servers (VSs)

□ Challenge/Problem

- With ***m*** VSs, each physical node has to maintain **$m \times \log N$** overlay connections

- Solution

 - Low cost virtual server selection

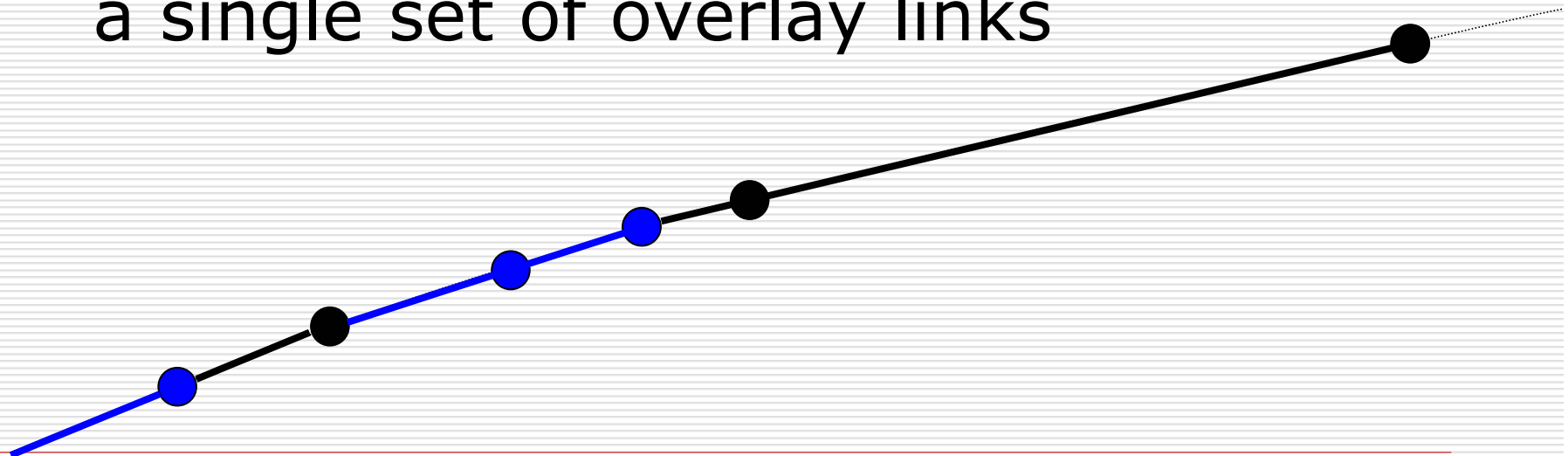
Low Cost Virtual Server Selection (LC-VSS)

□ Basic idea:

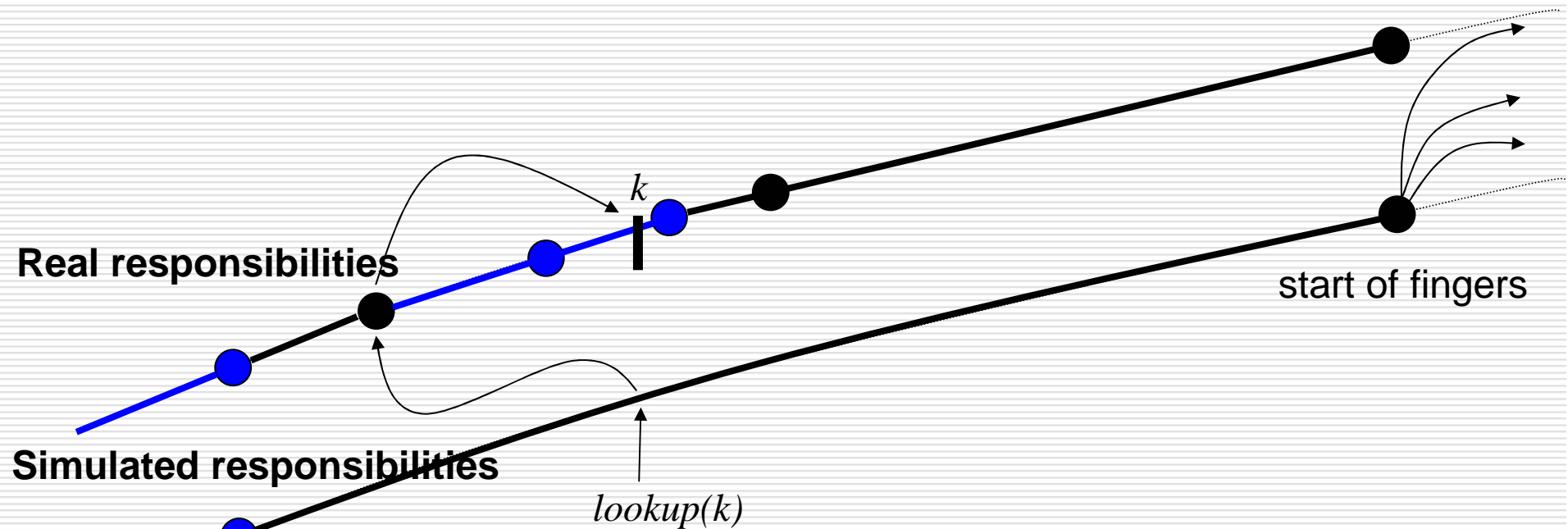
- Instead of picking VSs with random IDs, pick the IDs in a random fraction of the ID space

- Still responsible for non-contiguous regions

□ All VSs on a physical node can share a single set of overlay links



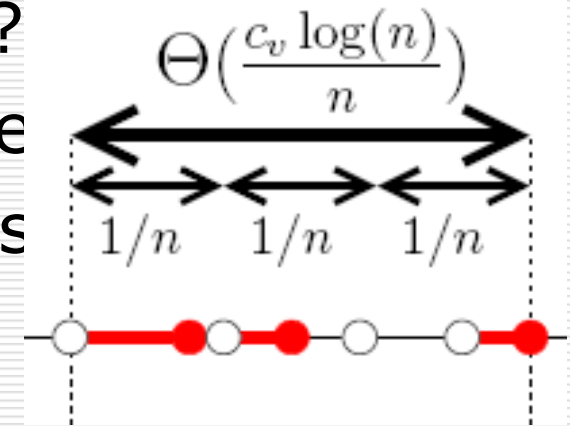
LC-VSS



- ❑ Simulate ownership of contiguous fraction
- ❑ Routing ends at simulated target
- ❑ Due to clustering, real owner is nearby in ID space, thus complete lookup using successor list

LC-VSS

- Which random fraction on ring?
- How long should the fraction be
- How far apart should the VS ids
- Answers: (network size = n)
 - Start at a random location
 - Size of random fraction $\Theta\left(\frac{c_v \log n}{n}\right)$
 - VS ids spaced at intervals apprx. equal to $\frac{1}{n}$
- Notice the fall?
 - Each node should know network size n
 - Interesting problem – solutions in the reading list

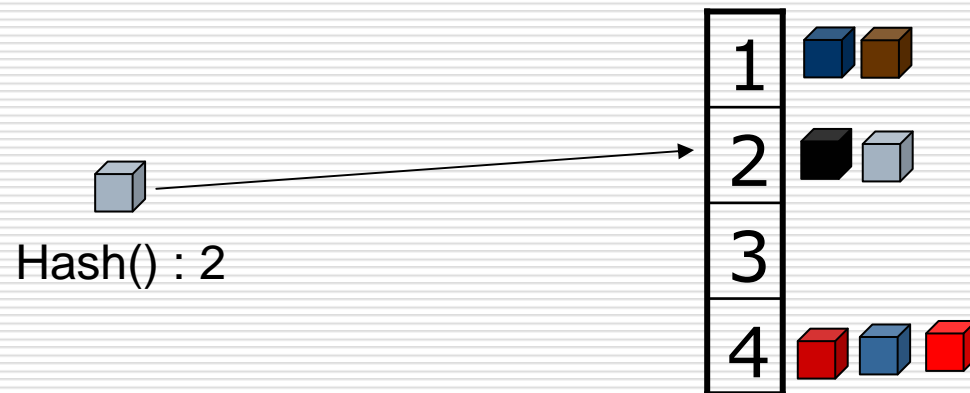


Overview

- ☑ Background
- ☑ The problem : load imbalance
- ☑ Causes of load imbalance
- ☐ Solutions
 - ☑ Method 1: Virtual servers
 - Method 2: Without virtual servers

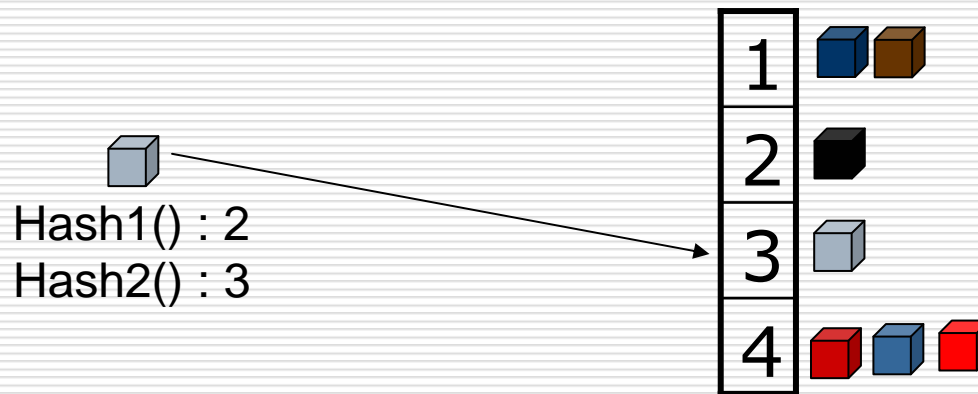
Standard Hash Table

- Hash function \rightarrow key is mapped to an entry in the table independently and uniformly at random
- In case of collision, all keys are stored in a link-list



Power of two choices

- Use multiple (here 2) hash functions
- Store at entry with shorter link-list



Amount of load balance

- Given n keys and a table of size n , length of longest link list
 - Standard hash table: $O(\log n / \log \log n)$
 - Power of two choices:
 - 2 hash functions: $= O(\log \log n / \log 2)$
 - $d \geq 2$ hash functions: $O(\log \log n / \log d)$
 - $d = \log n$: $O(1)$

Without virtual servers

- All nodes know the hash functions used, h_1, h_2, \dots, h_d
- To store an item
 - concurrent lookup peers responsible for each hash
 - store at node with minimum load
- To fetch an item
 - concurrent lookup peers responsible for each hash
 - one of the nodes will have the item stored

Hot Spots

- Replication
 - Consistency?
- Caching
 - Consistency?
- Part of reading list

References

- **Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications.** Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan. [IEEE/ACM Transactions on Networking \(TON\), 11\(1\):17–32, 2003.](#)
- **Heterogeneity and Load Balance in Distributed Hash Tables.** P. Brighten Godfrey and Ion Stoica. [INFOCOM 2004 + slides.](#)
- **Load Balancing in Dynamic Structured P2P Systems.** Brighten Godfrey, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, Ion Stoica. [INFOCOM 2004.](#)
- **Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems.** David R. Karger, Matthias Ruhl. [16th Annual ACM Symposium on Parallelism in Algorithms and Architectures.](#)
- **Simple Load Balancing for Distributed Hash Tables.** John W. Byers, Jeffrey Considine, Michael Mitzenmacher. [IPTPS 2003](#)