Load balancing mechanism in range query enabled P2P networks

2009. 08. 27

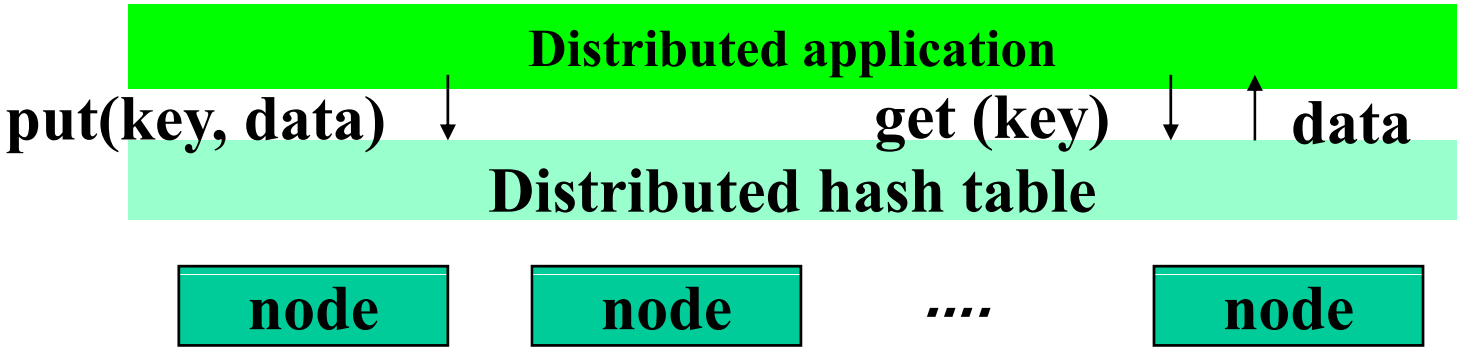
Park, Byunggyu

Contents

- Background
  - DHT(Distributed Hash Table)
- Motivation
- Proposed scheme
  - Compression based Hashing
  - Load balancing Technique using FRI(Fixed Routing Identifier)
- Summary

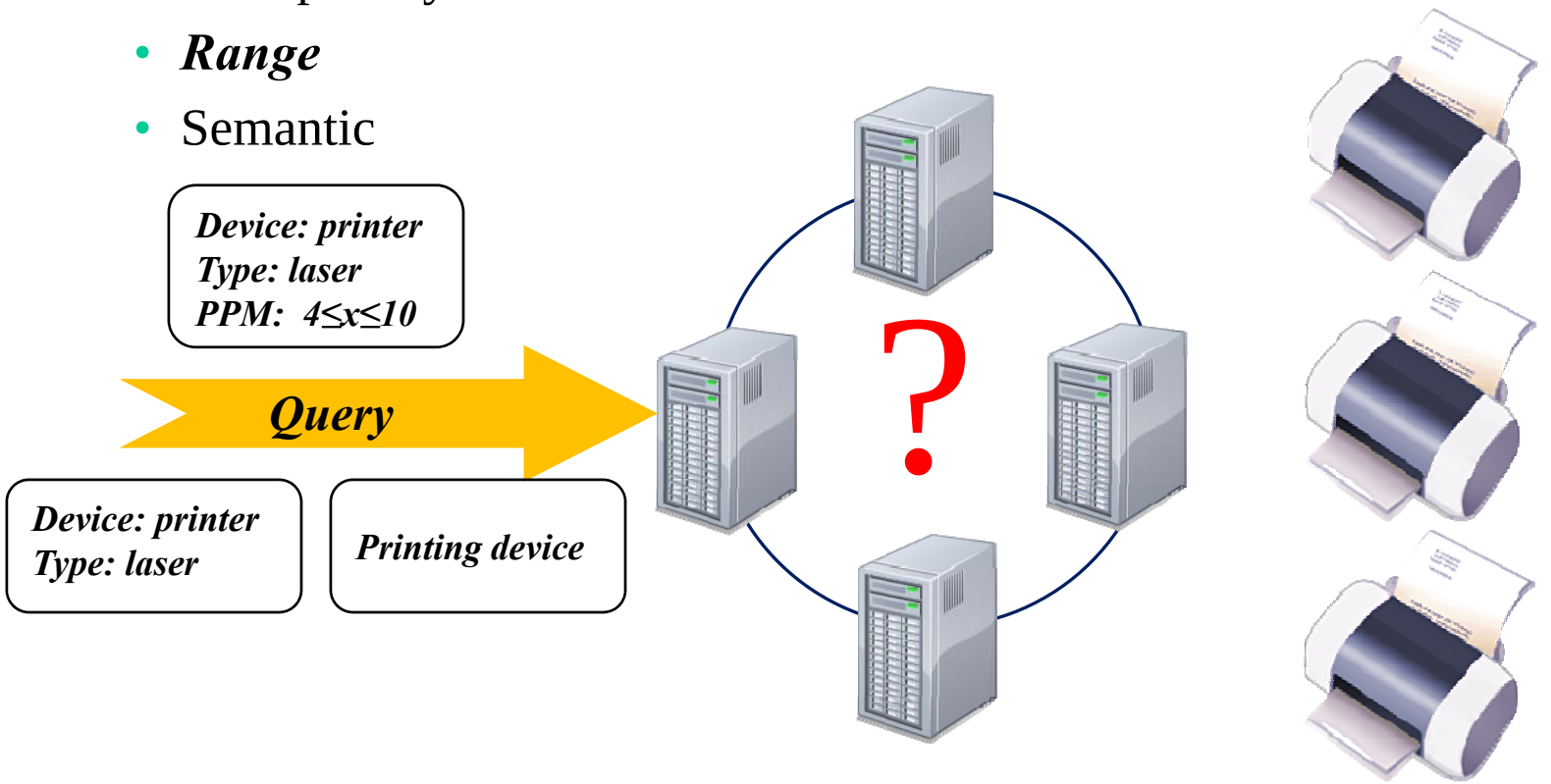
Background

- What is DHT?
  - DHT provides the object lookup service for P2P applications
    - Provide two primitives : ***put(k,v)***, ***get(k)***
    - Scalable
      - ***O(logN)*** *routing cost*
    - Provide load balancing
      - *Consistence hash function*: ***O(logN)*** imbalance
    - Support only *point query*
  - E.g. Chord, CAN, Pastry, Tapestry, etc



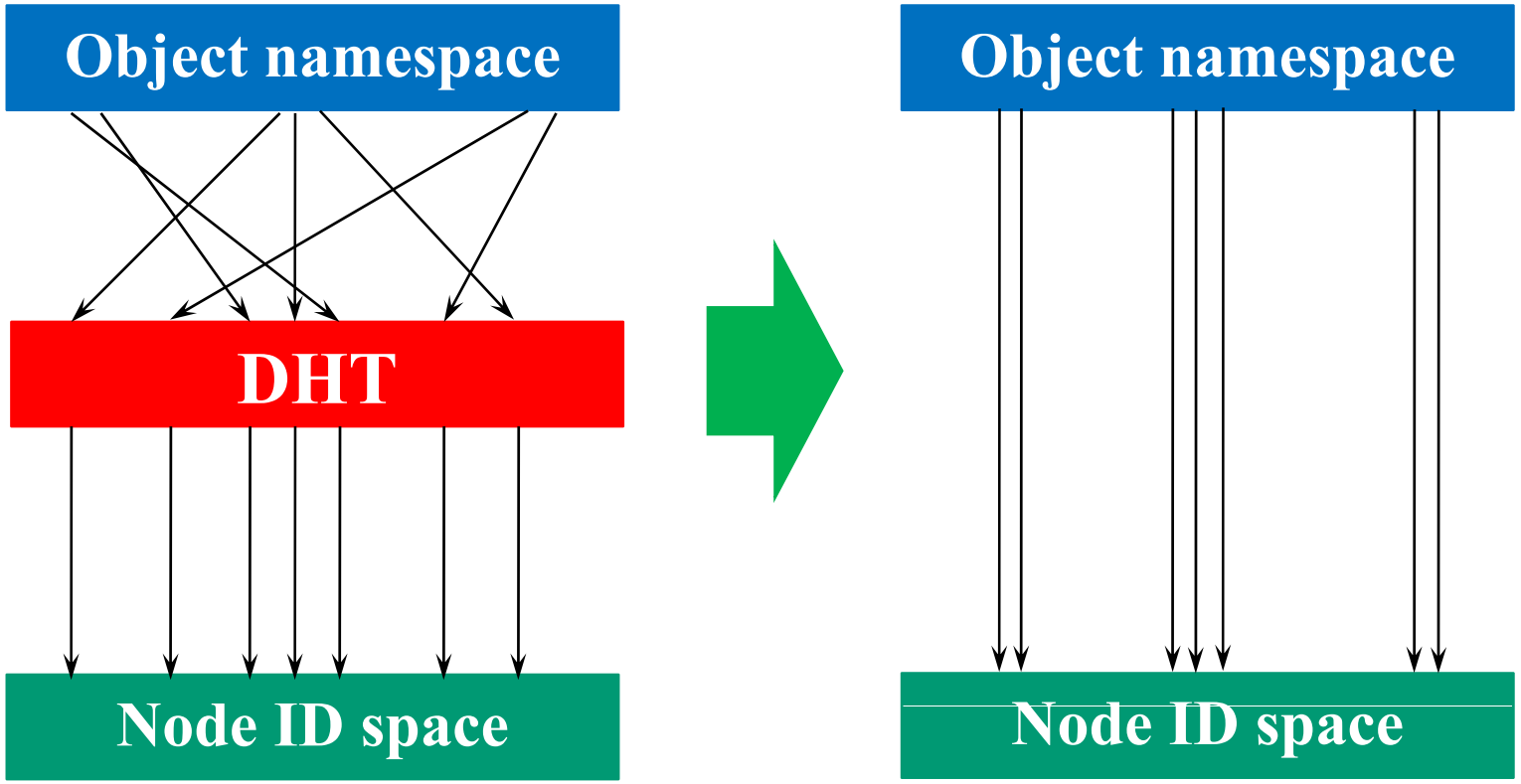
Background

- Due to ***un-flexible query support*** of DHT, application can be restricted
  - Currently many applications require ***complex query*** support
    - Multiple keyword
    - ***Range***
    - Semantic



Background

- How to enable flexible query support in DHT?

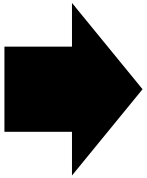


Lose *order semantics*
Support only *point query*
O(logN) Load imbalance

Order preserving mapping
Support *complex query*
Serious load imbalance

Motivation

1) Order-preserving mapping can provide ***flexible query***
2) It causes ***load imbalance problem*** because of clustered property of data

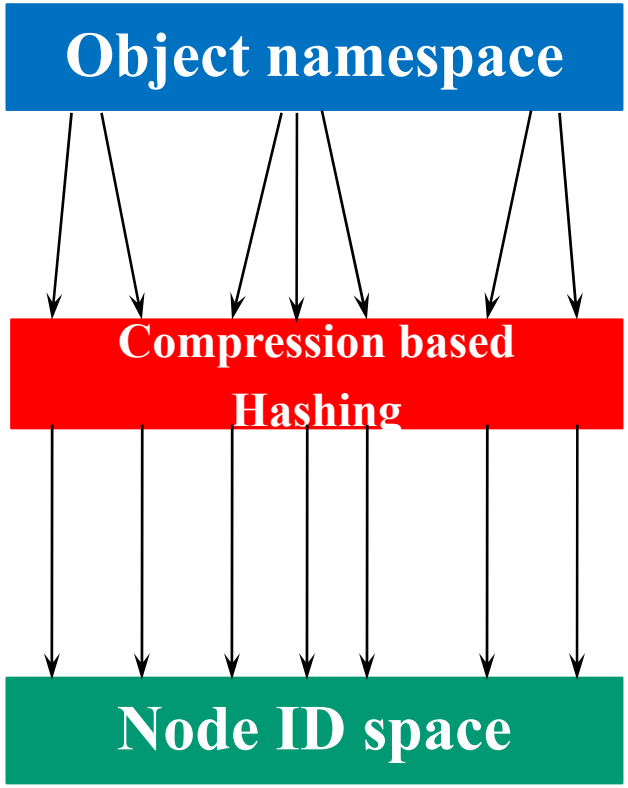


How to provide flexible query in DHT with balancing load?

Proposed Scheme

- Compression based Hashing
  - Based on arithmetic coding
  - Order-preserving mapping balancing load
- Load balancing Technique using FRI(Fixed Routing Identifier)
  - Based on virtual server

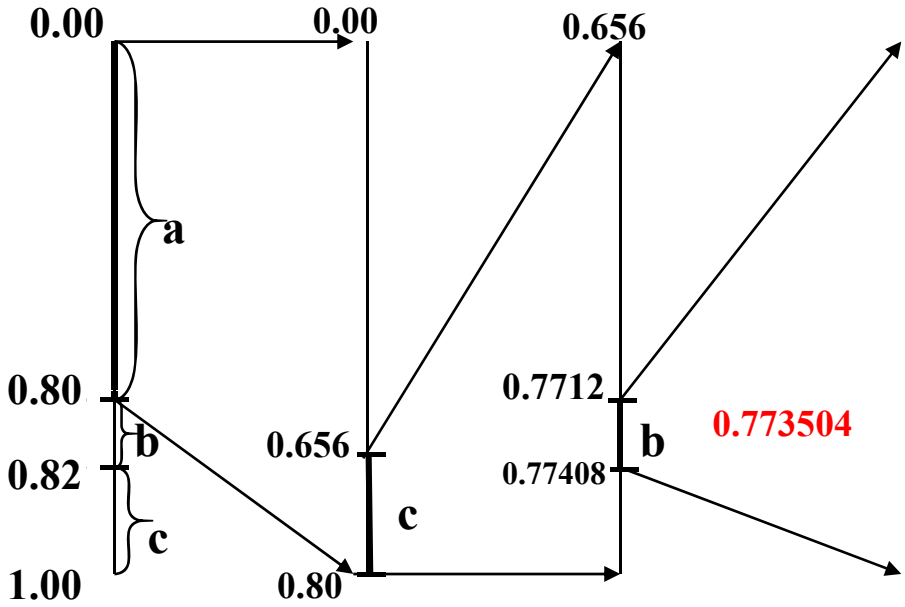
Compression based hashing



***Order preserving* mapping**
Support *complex query*
Relaxed load imbalance

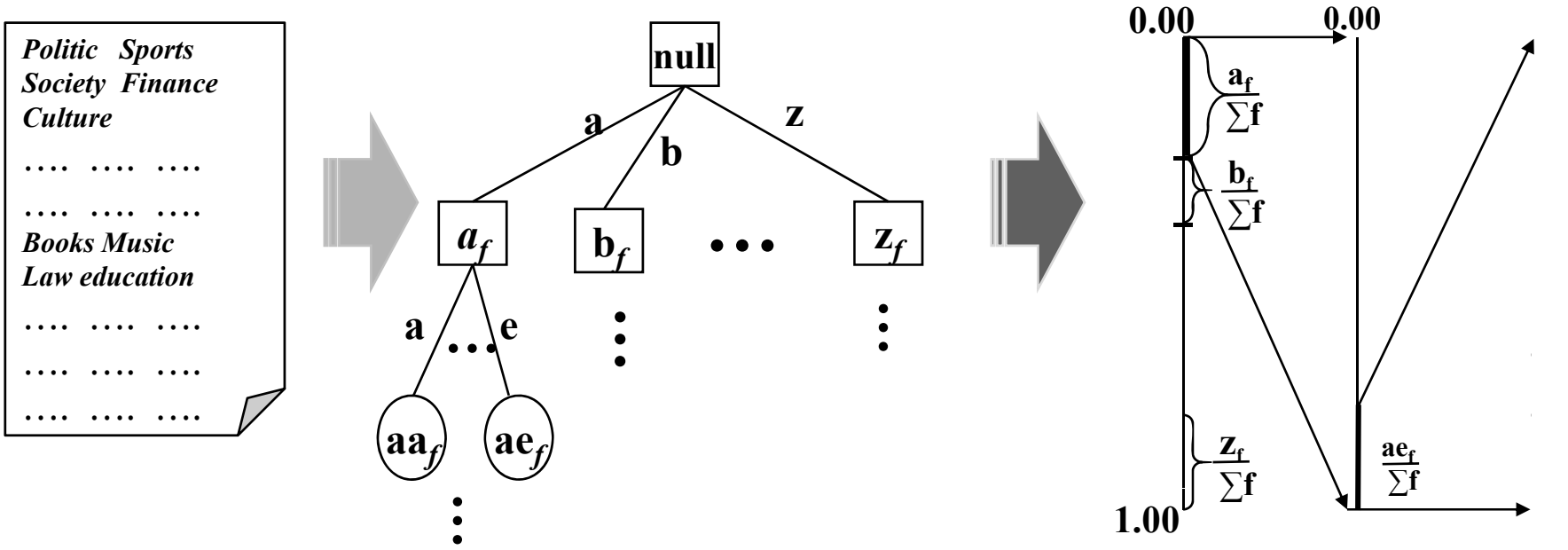
Symbol	probability	Range
a	0.80	[0.00, 0.80)
b	0.02	[0.80, 0.82)
c	0.18	[0.82, 1.00)

acb ≈ 0.773504



<Arithmetic coding>

Compression based Hashing



Sample data of corpus

Construct *trie* and calculate *frequency* of each symbol

Get the *compressed value* using arithmetic coding

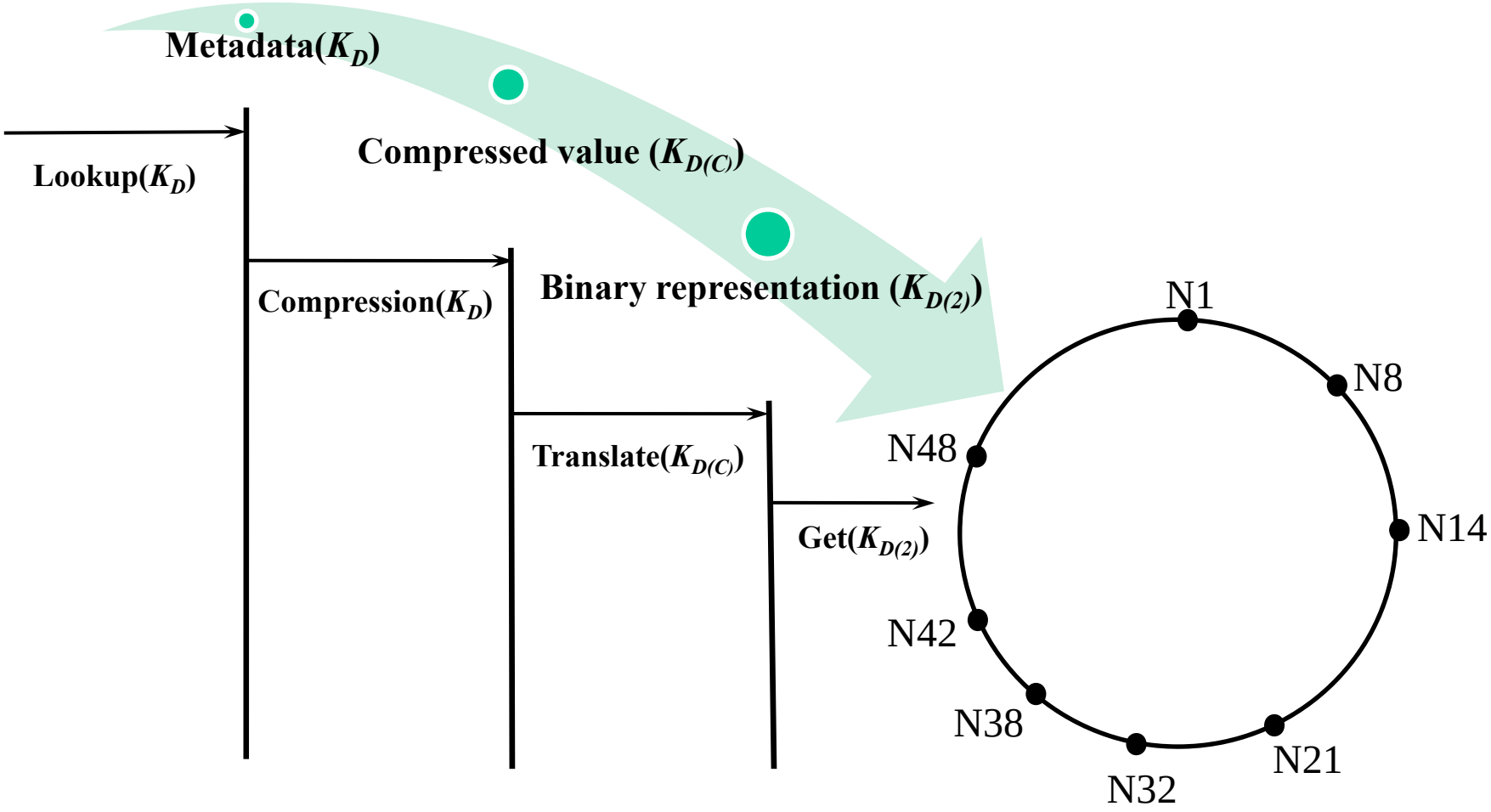
System pre-processing

Peer processing

Binary representation **of compressed value**

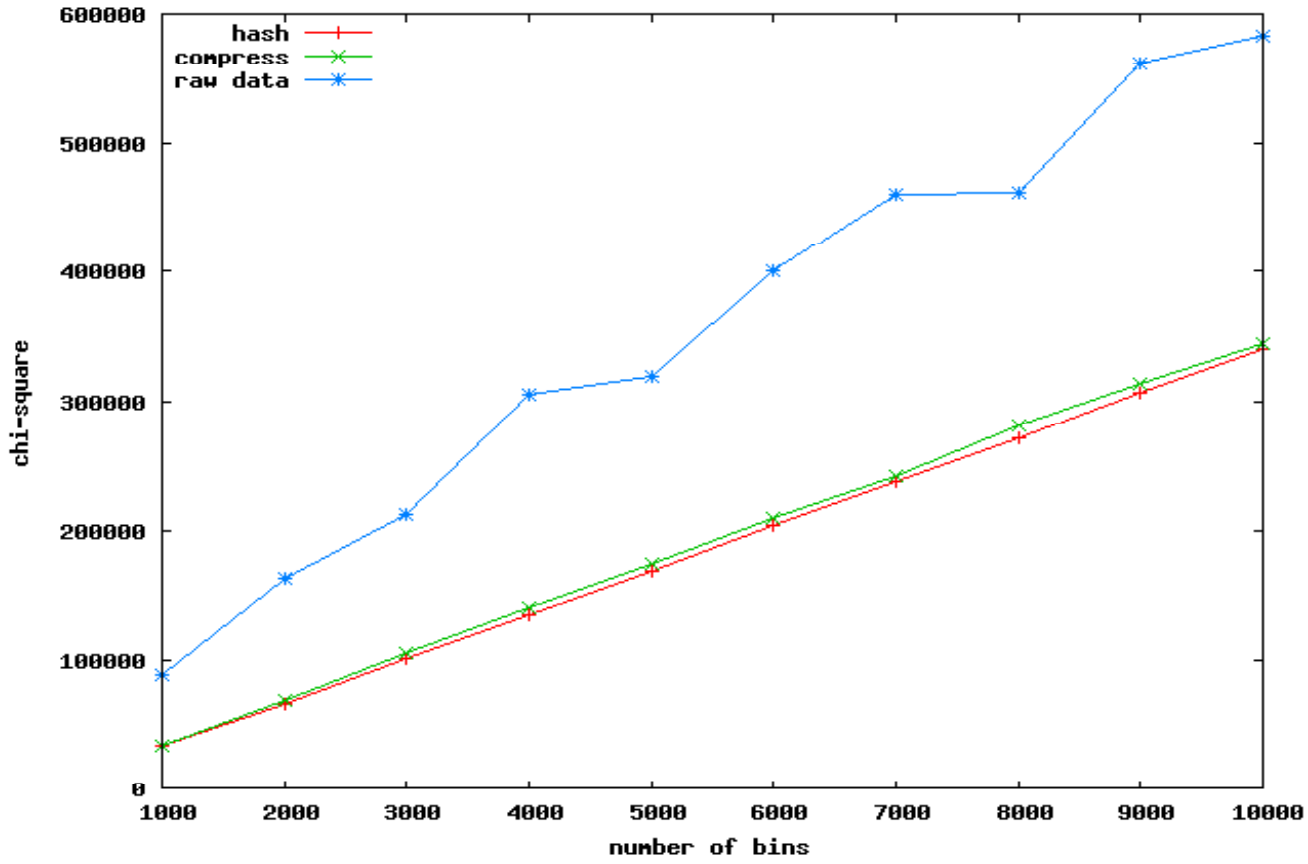
Compression based Hashing

- Lookup process



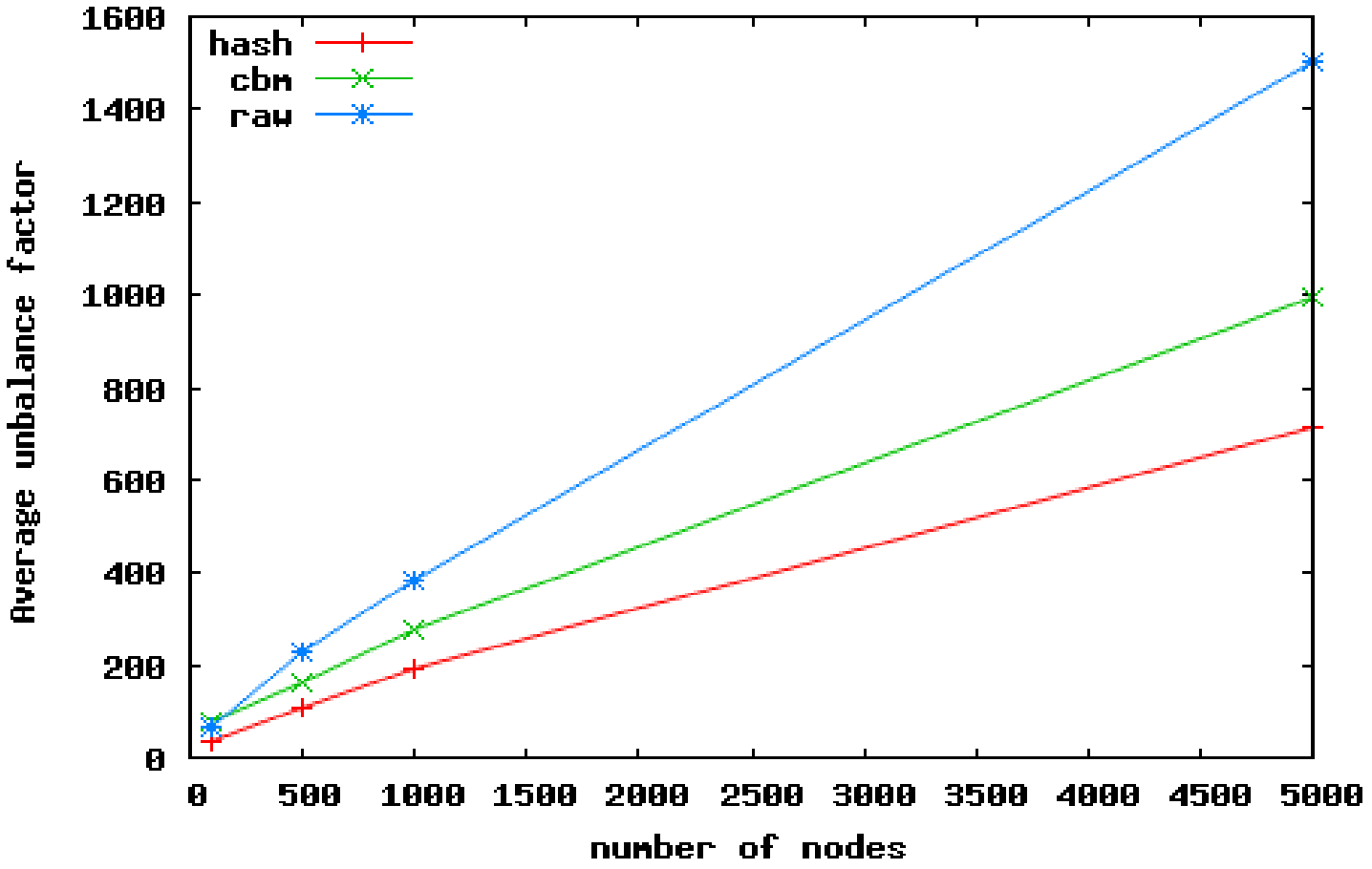
Evaluation

- Data uniformity
  - Training data set : ACM keyword



Evaluation

- Unbalance factor $\{(L_i - E)^2/E\}$ of brown corpus
  - Training data set: ACM keyword



Compression based Hashing

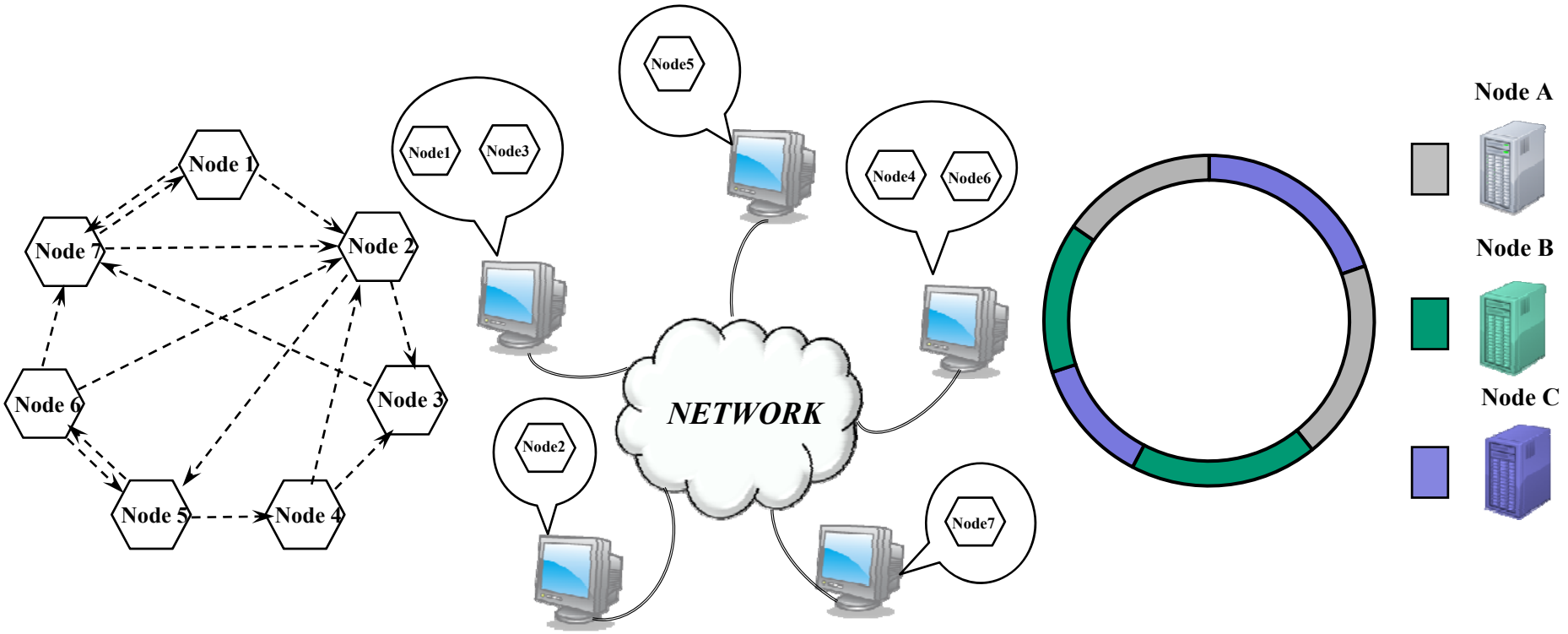
- Advantage of CBH
  - Obtain uniform data distribution
  - Order-preserving mapping
    - Support complex query
  - Flexible
    - Can be applied to different type of data model
- Dis-advantage
  - Require training data set
  - Performance depends on accuracy of sample data

Load balancing Techniques in DHT

- Selective node join
- Node migration
  - Support ***dynamic*** load balancing
  - Virtual server
    - Provide ***find-grained*** load balancing
    - No change in underlying DHT
    - **High maintenance cost**
      - ***O(logN)*** *virtual servers* per physical node
      - ***O(LogN)²*** *routing entries* per node
    - **Long query routing length**
    - **Unstable routing**
  - Dynamic node leave/join
    - Simple
    - Hard to provide *fine-grained* load balancing
- Replication/Cache

Load balancing Techniques in DHT

- Virtual server
  - Logical node in DHT
  - Transfer unit in order to balance load



<Logical view of V.S >

<Physical view of V.S>

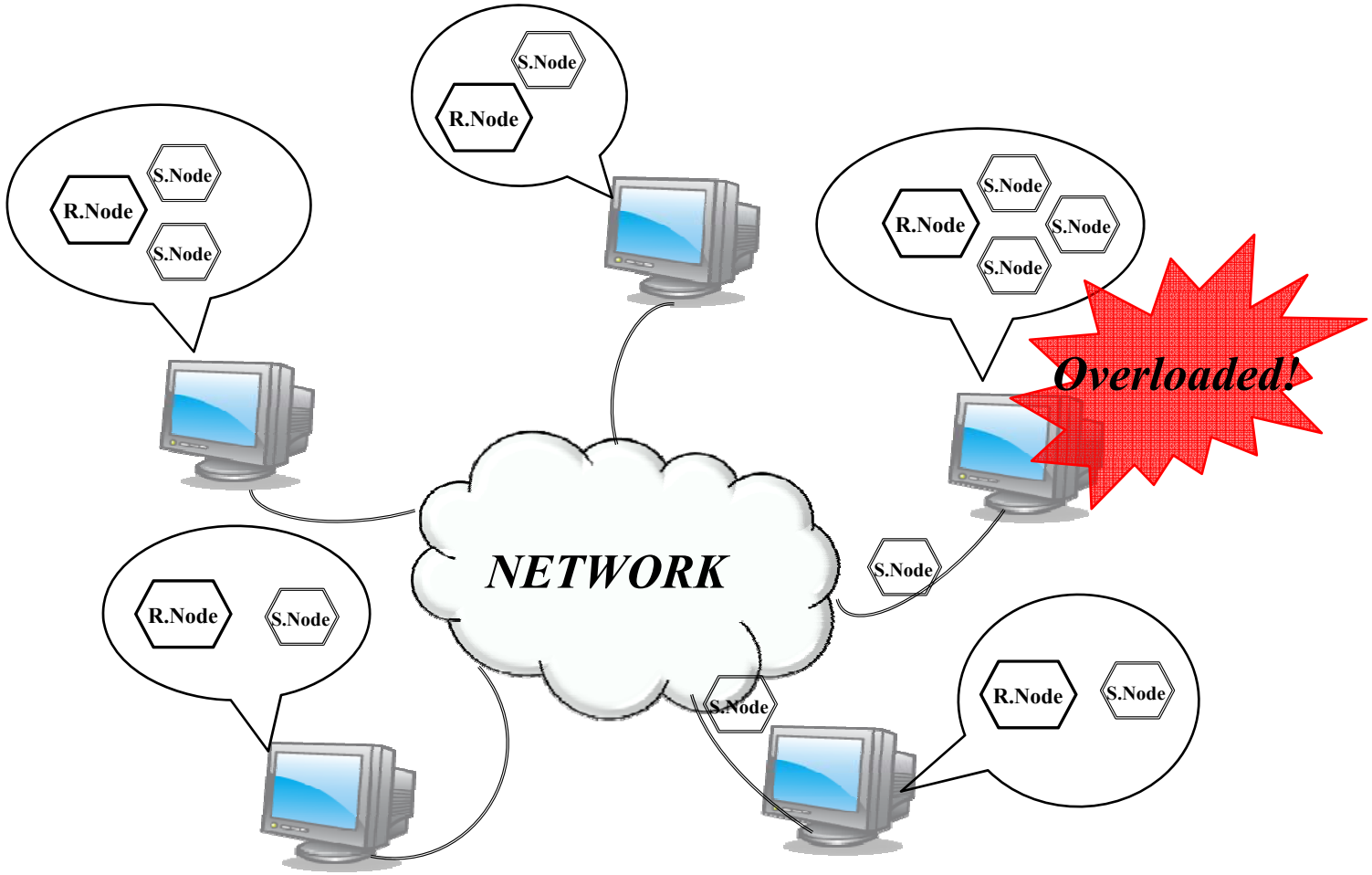
<Chord with V.S >

Load balancing Technique using FRI

- Node is classified into two types of nodes
  - One physical node has one ***routing node*** and several ***storage nodes***
    - Routing node
      - Has *fixed routing identifier*
      - Maintains ***O(logN)*** *routing entry*
    - Storage node(virtual server)
      - Has *storage identifier* and shares *routing identifier*
      - Maintains *constant number* of routing entry
      - Predecessor of routing node + successor of routing node
      - Can be migrated to other node

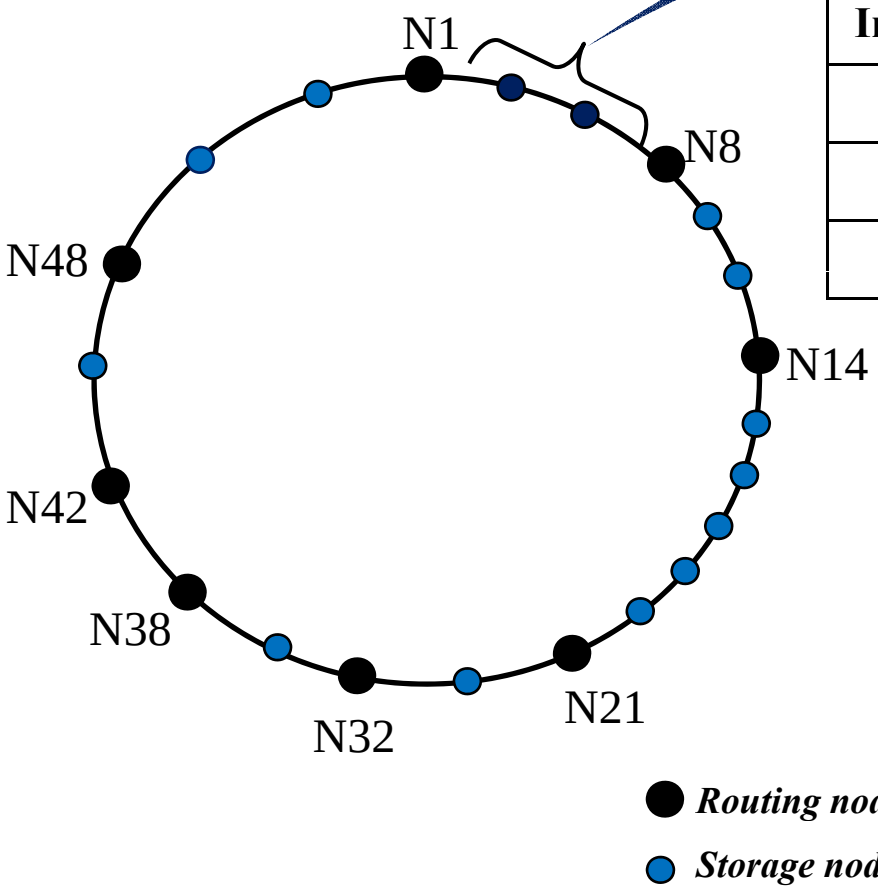
Load balancing Technique using FRI

- Load balancing based on V.S



Routing in Chord using FRI

- Routing table structure



Interval	F.R.I
1	N8
2	N8
…	…

Nodes	S.R.I
$Snode_0$	$S.R.I_0$
$Snode_1$	$S.R.I_1$
…	…

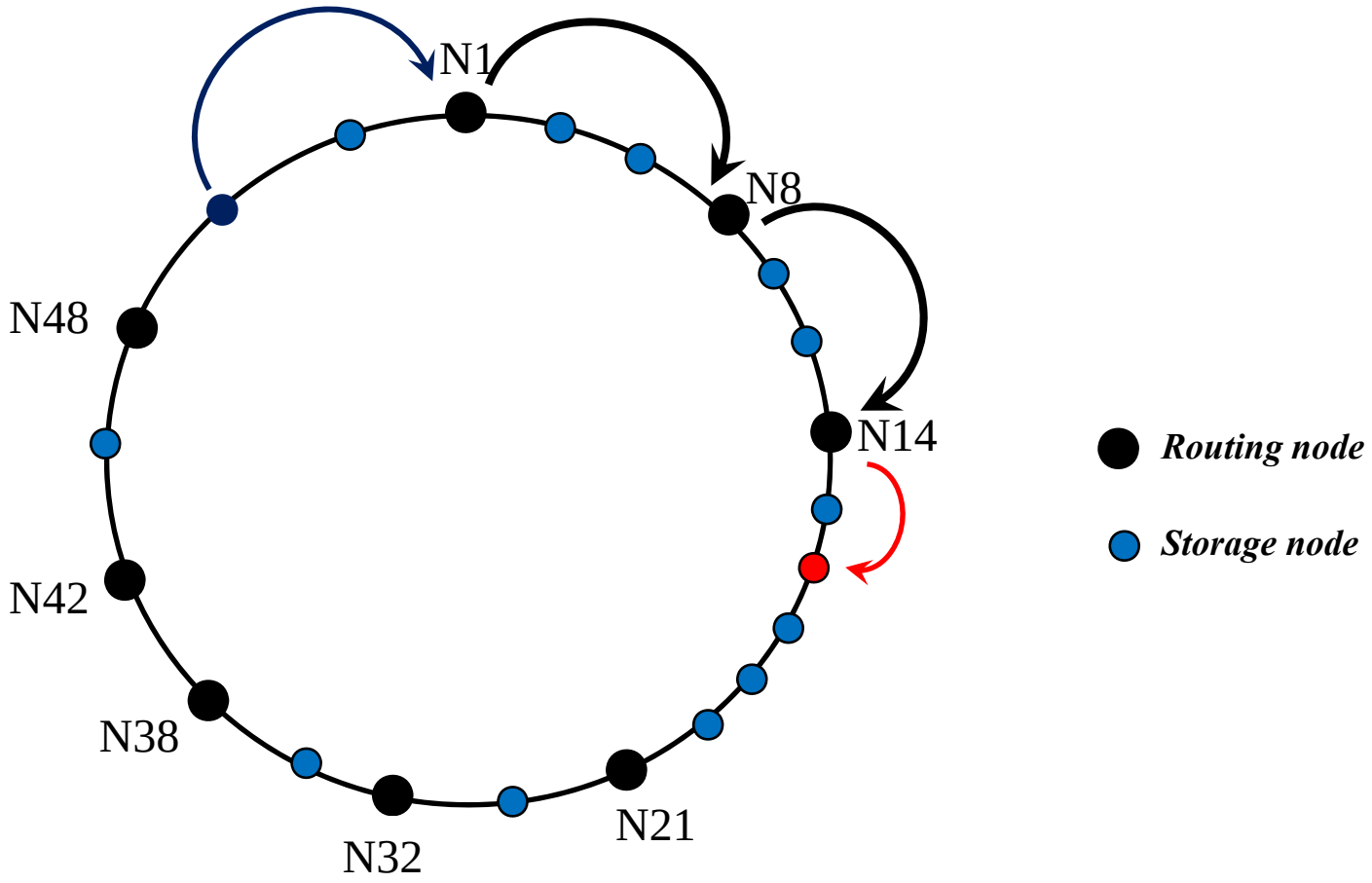
<Routing table of Routing node N1>

Type	F.R.I
Shared F.R.I	N1
$Successor_R$	N8
-	-

<Routing table of Storage node>

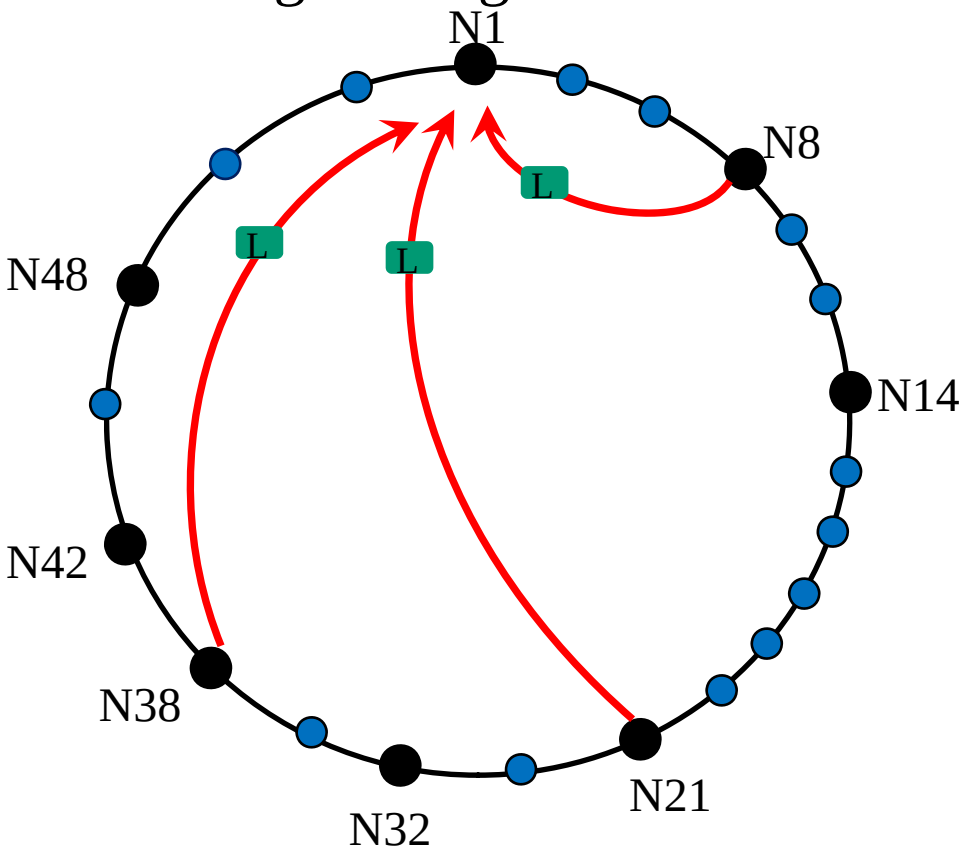
Routing in Chord using FRI

- Routing example



Load balancing in Chord using FRI

- Load information gathering
  - Log(N) information from finger table
  - Used to reassign storage node



Interval	F.R.I
1	N8
2	N8
…	…

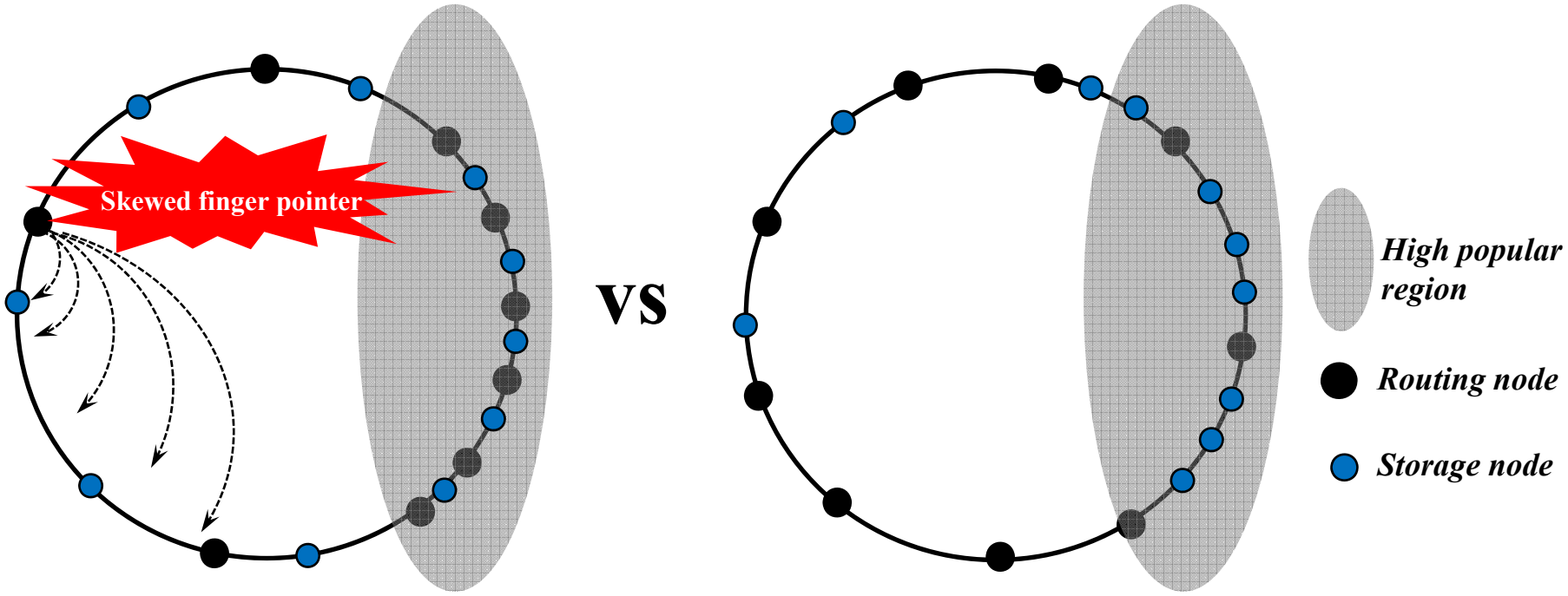
Finger table of N1

Routing node

Storage node

Load balancing in Chord using FRI

- Clustered routing node vs uniform routing node



Can not guarantee *logN* routing hops

Hard to get overall load information

Guarantee *logN* routing hops

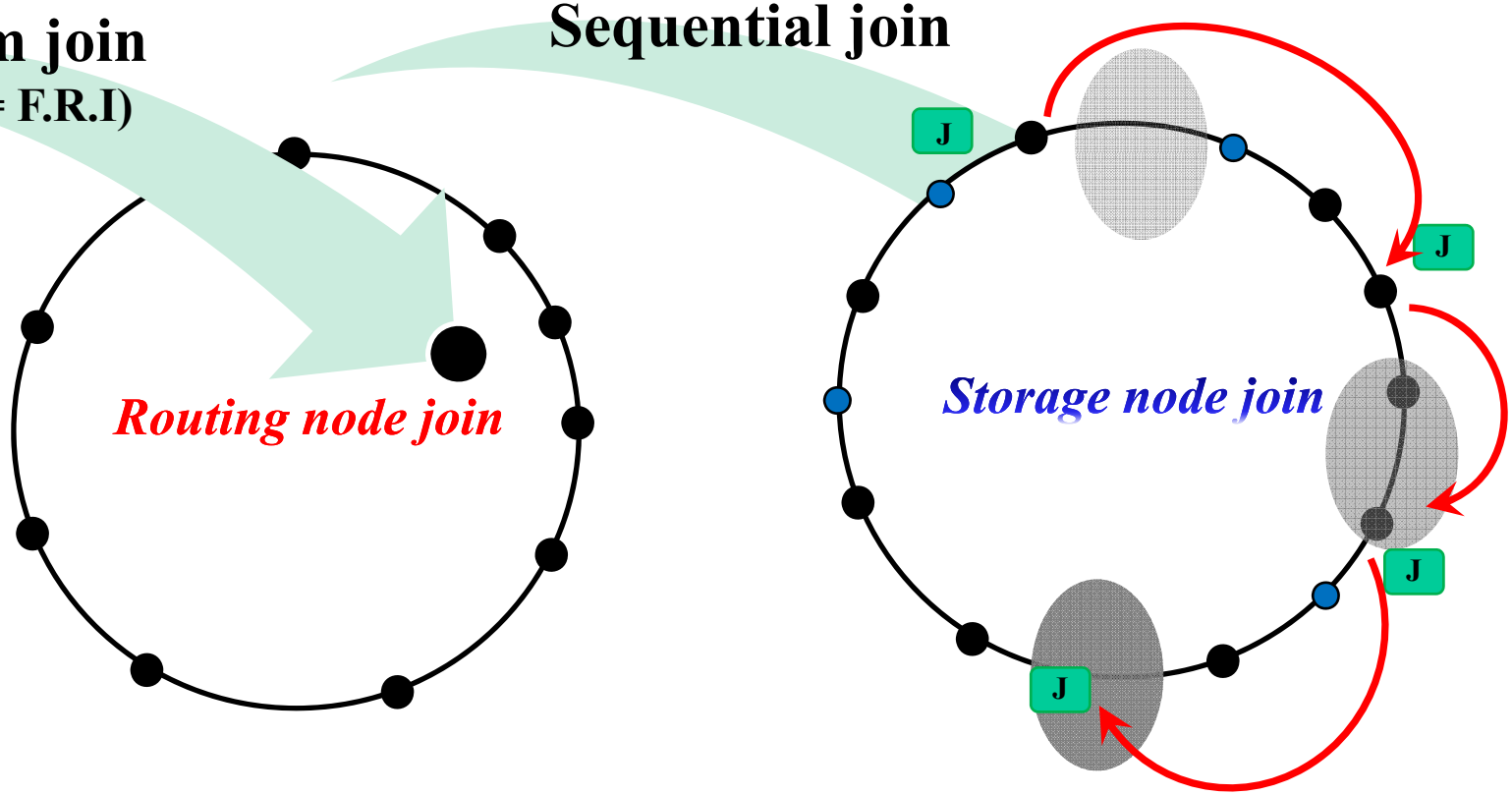
Easy to get overall load information

Load balancing in Chord using FRI

- Node join

Random join
(h_c(IP) = F.R.I)

Sequential join



Uniform routing node distribution
Efficient load sampling
Optimal routing

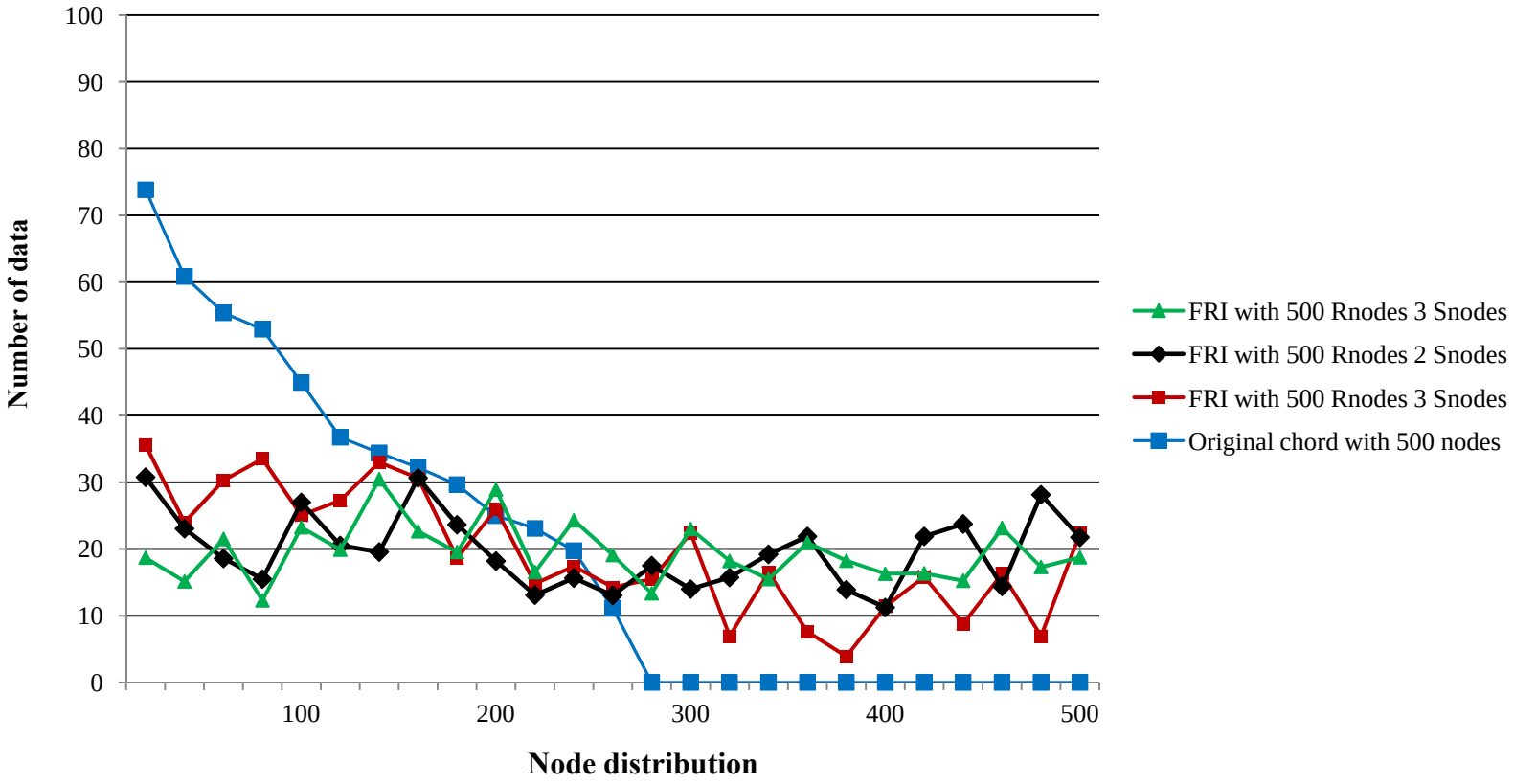
Sequential join to find heavy region
Skewed storage node distribution
Fine-grained load balancing

Load balancing Technique using FRI

- Advantage of FRI
  - Include all advantages of virtual server scheme
    - Fine-grained load balancing
    - General and flexible
  - Solve inherent problems of virtual server
    - Still provide *O(logN)* maintenance cost per physical node
    - Stable routing
    - Shorter query routing path
- Dis-advantage
  - Change in underlying DHT
    - Routing algorithm

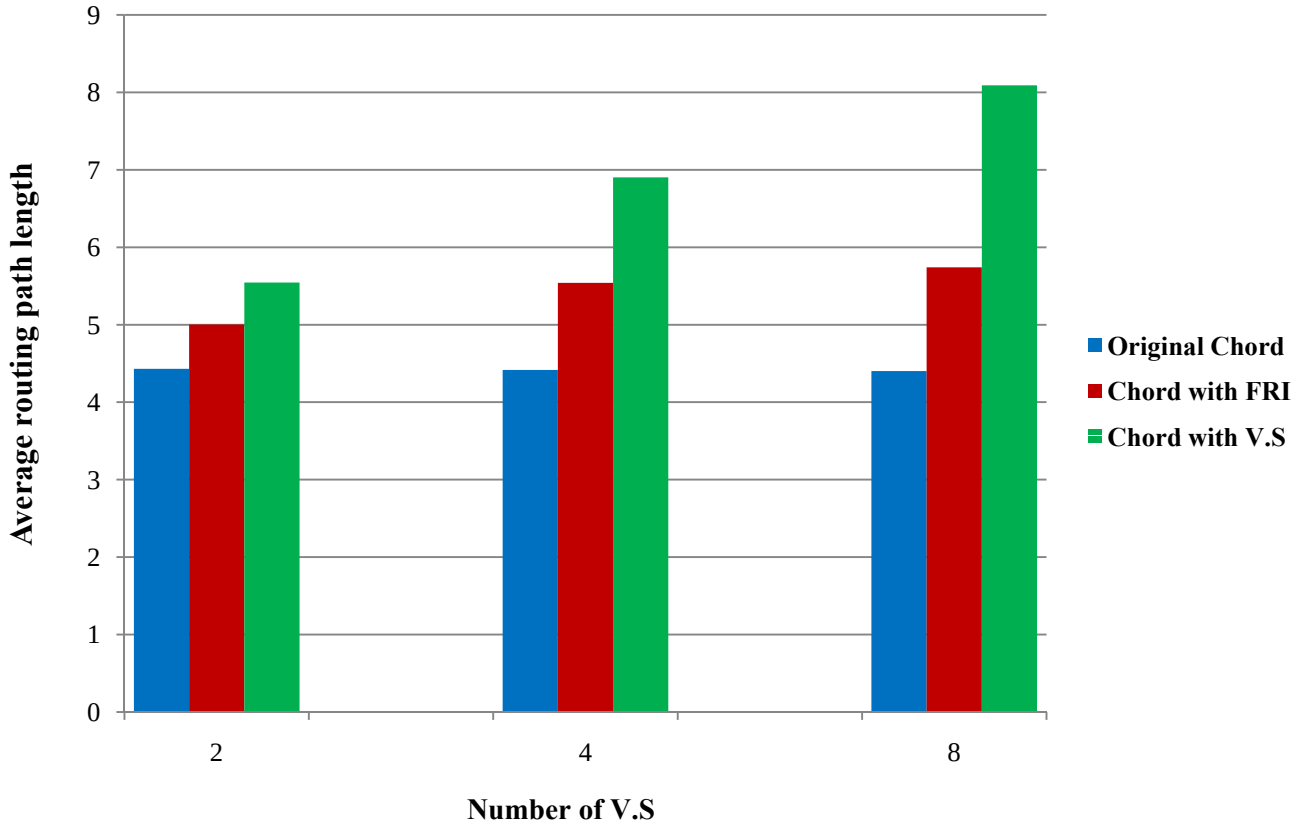
Evaluation

- Load distribution according to increase of number of storage nodes



Evaluation

- Average routing path length of FRI vs Virtual Server



Summary

- To satisfy various application requirement, complex query should be supported in DHT layer
  - Order-preserving mapping
  - Load balancing problem due to skewed data distribution
- Compression based hashing
  - Provide order-preserving mapping
  - Provide uniform data distribution
- Load balancing Technique using FRI
  - Dynamic load balancing based on virtual server
  - Reduce maintenance overhead
    - ***O(logN)*** + *C* routing entries
  - Fine-grained balancing
  - Shorter routing path

Future Work

- C.B.H
  - How to guarantee the accuracy of sample data?
  - Comparison with other order-preserving hashing function

- Load Balancing using F.R.I
  - Comparison with V.S scheme
    - Balancing ratio(number of nodes, storage nodes, data…)
    - Balancing overhead(dynamicity of P2P...)
    - Maintenance cost

Thank you