

University of Maryland, College Park
CMSC818K, Spring 2009

Pastry & PAST

Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems
Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility
by Antony Rowstron (Microsoft Research, Cambridge, UK) and Peter Druschel (Rice University)

February 24, 2009
Sukhyun Song

Outline

2

□ Pastry

- Node State (Routing Table, Leaf Set, Neighborhood Set)
- Routing Algorithm
- Self-organization
- **Locality Property**
- Discussion

□ PAST

- **Storage Management** (Replica Diversion, File Diversion)
- Caching
- Discussion

What is Pastry?

3

A scalable, distributed object location and routing substrate for wide-area peer-to-peer applications

- Pastry creates a **self-organizing overlay network**.
- Pastry provides an **efficient routing** capability.
 - ▣ It routes a message to the node numerically closest to a key.
 - ▣ $O(\log N)$ routing steps
- Pastry takes into account **network locality**.
 - ▣ It seeks to minimize the distance messages travel.
 - ▣ A message first reaches the node physically closest to the client among k nodes.

Node Identifier

4

- **128-bit nodeid** and key
 - ▣ A sequence of digits **with base 2^b** (for the purpose of routing)
 - b is normally set to 4.

- Randomly assigned when a node joins
 - ▣ A cryptographic hash of the node's public key or IP address

- **Uniformly distributed** in the circular nodeid space.
 - ▣ Nodes with numerically adjacent nodeids are widely dispersed in underlying physical network.

Node State – Routing Table (R)

5

- Size: $\lceil \log_{2^b} N \rceil$ rows \times $2^b - 1$ columns
- An entry at row r and column c
 - Shares the present nodeld in the first r digits (*common prefix*)
 - Has c as the value of its $r+1^{\text{th}}$ digit (*next digit*)
- Trade-off between the size and the number of hops
 - Bigger $b \rightarrow$ Smaller routing hops (# rows) but bigger routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Nodeld 10233102

Example of $b = 2$, 8 digits

\rightarrow 16-bit nodeld space

Entry (3,2)

Shares the first 3 digits (102)

Has 2 as the fourth digit value

Node State – Leaf Set (L)

6

- Size: $|L|$ (typically 2^b or 2^{b+1})
- Nodes with numerically closest nodeIds
 - ▣ $|L|/2$ larger nodeIds and $|L|/2$ smaller nodeIds

Example of $b = 2$, 8 digits \rightarrow 16-bit nodeId space

Leaf set	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

NodeId 10233102

Node State – Neighborhood Set (M)

7

- Size: $|M|$ (typically 2^b or 2^{b+1})
- Physically closest nodes
 - ▣ According the proximity metric
- Useful in maintaining locality properties
 - ▣ Normally not used in routing messages

Example of $b = 2$, 8 digits \rightarrow 16-bit nodeid space

Neighborhood set			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Nodeid 10233102

Routing – Algorithm

8

- Goal: Route a message to a node with numerically closest nodeId to a key in $O(\log N)$ steps

- First case (one step): The key is within the range of L
 - ▣ Forward to the closest node in L
 - ▣ If I'm the closest node to the key, the routing is done!

- Second case ($\log N$ steps): The entry in R exists
 - ▣ Forward to the node at the entry in R
 - ▣ The key is getting closer to the closest nodeId by one digit.

- Third case (one step): Otherwise
 - ▣ Forward to any closer node in L U R U M (mostly picked from L)
 - ▣ Rare case

Routing – Example

9

- The range of L: 10233000 ~ 10233232

NodeId 10233102			
Leaf set	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232
Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	
Neighborhood set			
13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

- 10233005 → 10233001
 - First case: in L
- 10233250 → 10233-2-32
 - Second case: (5,2) in R
- 10233300 → 10233232
 - Third case: in L
 - (5,3) in R is empty.

Self-organization – Node Join

10

- ❑ $X \rightarrow A \rightarrow B \rightarrow C \rightarrow \dots \rightarrow Z$: the “join” request
 - ❑ **X: new node**
 - ❑ **A: physically closest node to X**
 - ❑ Z: numerically closest node to X
 - ❑ $(A \sim Z) \rightarrow X$: node state tables to construct X’s table
 - ❑ $A \rightarrow X$: neighborhood set, A_0 (row 0 entries in R of A)
 - ❑ $B \rightarrow X$: B_1 (B and X share one digit)
 - ❑ $C \rightarrow X$: C_2 (C and X share two digits)
 - ❑ ...
 - ❑ $Z \rightarrow X$: leaf set
 - ❑ $X \rightarrow (A \sim Z)$: a node state table to update A~Z’s tables
- $O(\log N)$ messages



for the locality property in the routing table

Self-organization – Node Departure

11

- Repair L (leaf set)
 - ▣ Node failure detected by heartbeat monitoring
 - ▣ Contact the smallest or largest node in L and update L

- Repair M (neighborhood set)
 - ▣ Node failure detected by heartbeat monitoring
 - ▣ Query all the other nodes in M and update M

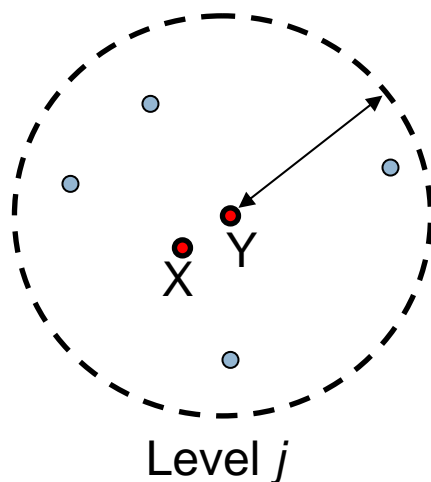
- Repair R (routing table)
 - ▣ Invalid routing entries detected when attempting to route
 - ▣ Query the other nodes in row for replacing the failed entry
 - ▣ Query successive rows until success

Locality Property – Routing Table (1)

12

Each entry in R is physically closest to the present node among appropriate nodes for the entry.

- Assume every node's R satisfies the locality property.
- In level j (between two nodes sharing the first j digits)
 - ▣ If a new node X knows the physically closest node Y in level j , **X can take Y_j to fill in X_j** to satisfy the locality property. (with a little error by moving center)



radius: the expected average physical distance between two nodes in level j

Locality Property – Routing Table (2)

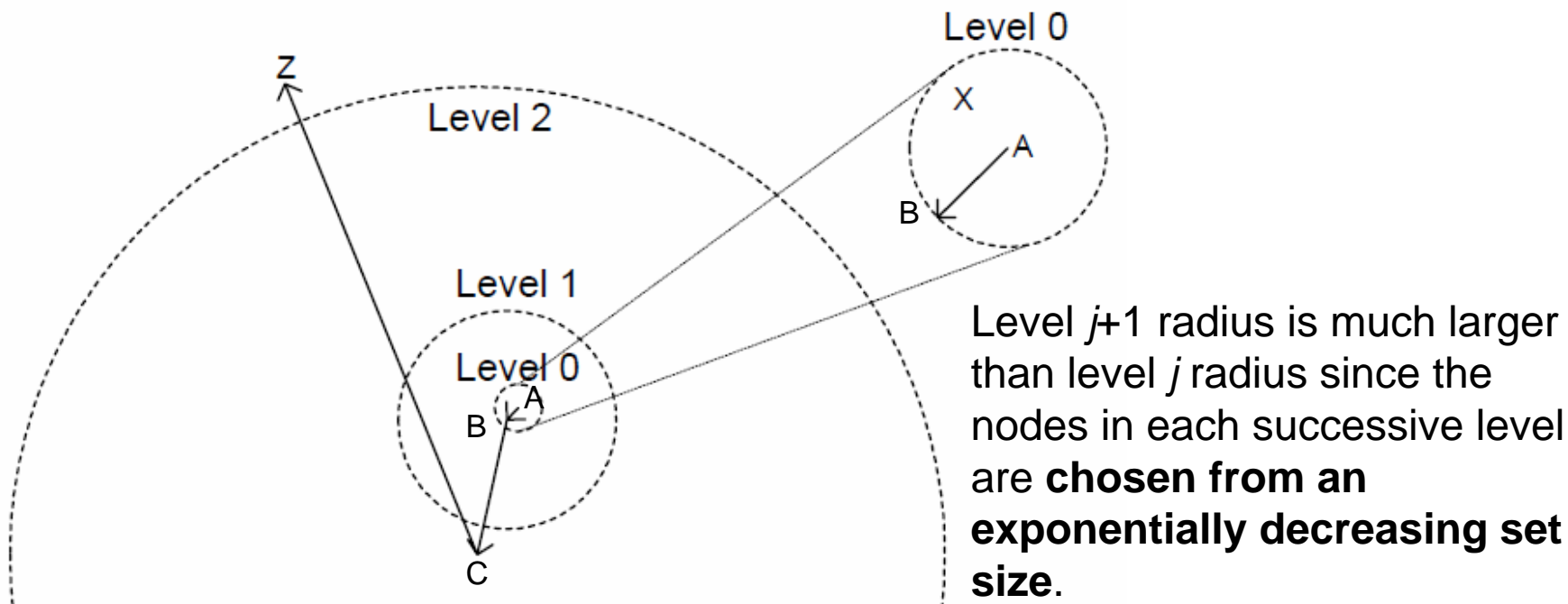
13

□ In all levels

- $X \sim A$ in level 0. X takes A_0 . $A \sim B$ in level 0.
- $X \sim B$ in level 1. X takes B_1 . $B \sim C$ in level 1.
- $X \sim C$ in level 2. X takes C_2 . $C \sim D$ in level 2.
- ...



So R of X satisfies the locality property.

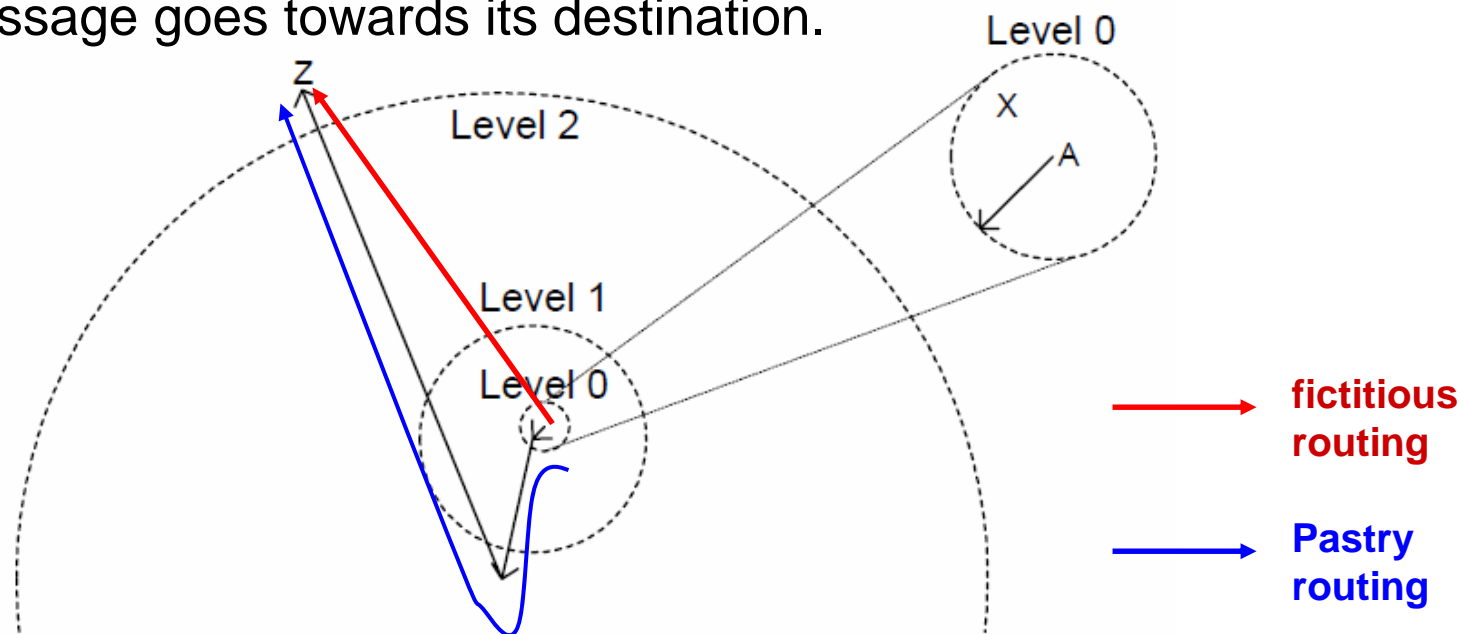


Locality Property – Short Routing Path

14

A message routes on a globally short path.

- ▣ The locally closest node is picked as the next hop in each level.
 - ▣ The expected distance to the next hop at each routing step is **exponentially increasing**.
- So, the message tends to make larger strides **with no possibility of returning** to the previous node. Therefore, the message goes towards its destination.



Locality Property - the Nearest among k Nodes

15

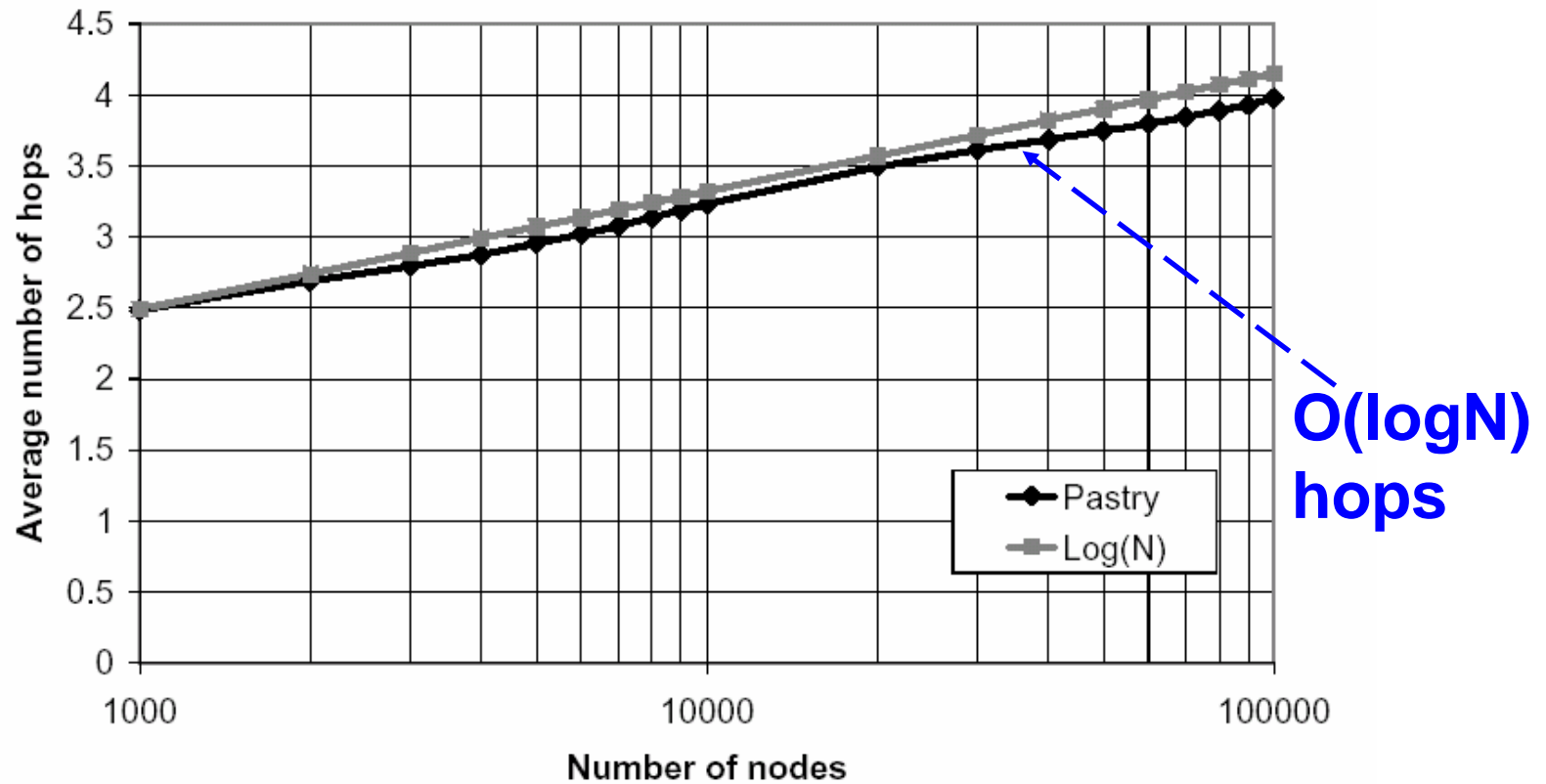
When k replicas are already located in numerically closest k nodes to a key, a client can first reach the physically closest node among the k nodes.

- With a heuristic based on estimating the density of nodes
 - ▣ Detects when a message approaches the set of k numerically closest nodes
 - ▣ Then, switches to numerically nearest address based routing

Result - # Routing Hops

16

- The number of routing hops = $\log(N)$

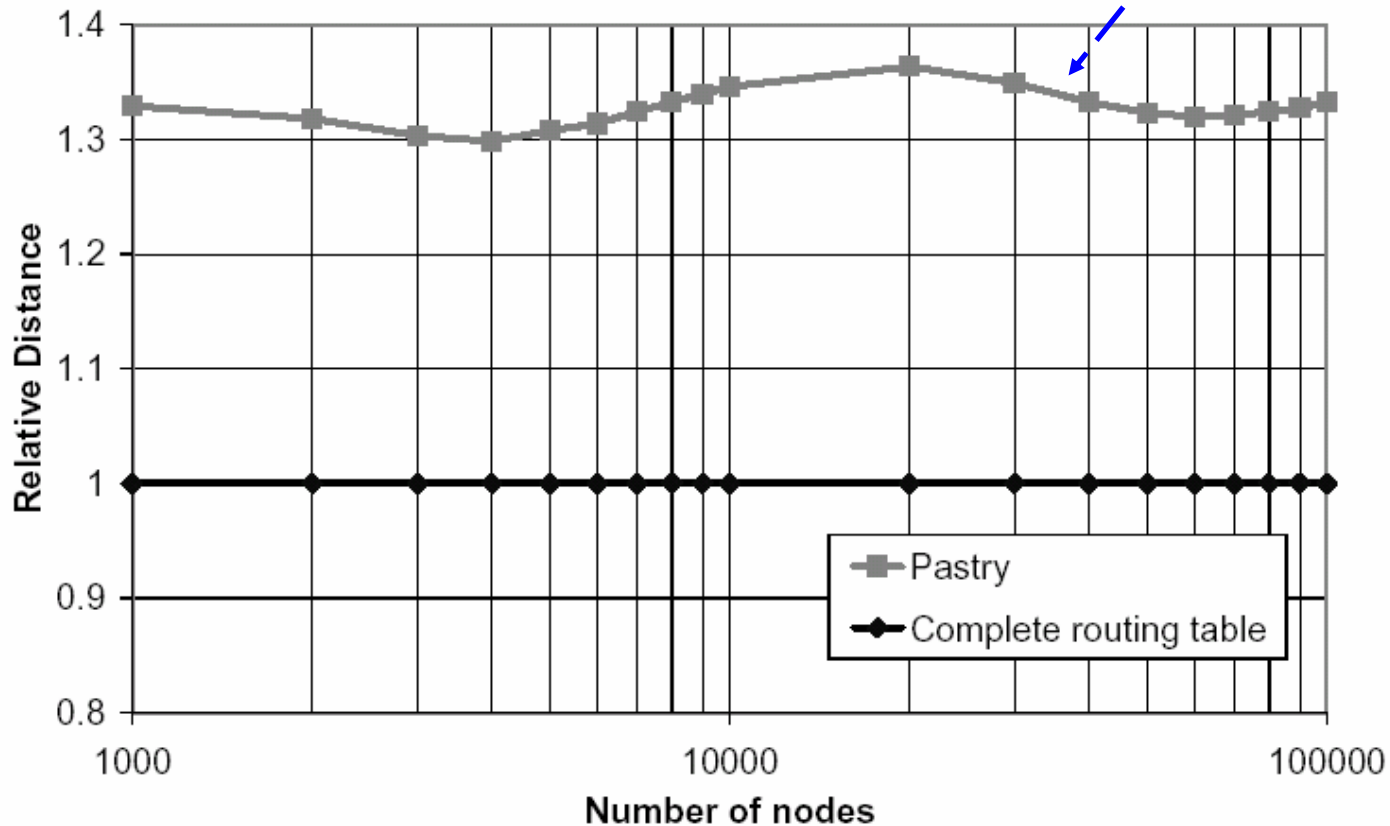


Result – Globally Short Routing

17

- Fictitious routing distance (1)
- Pastry routing distance (1.3X)

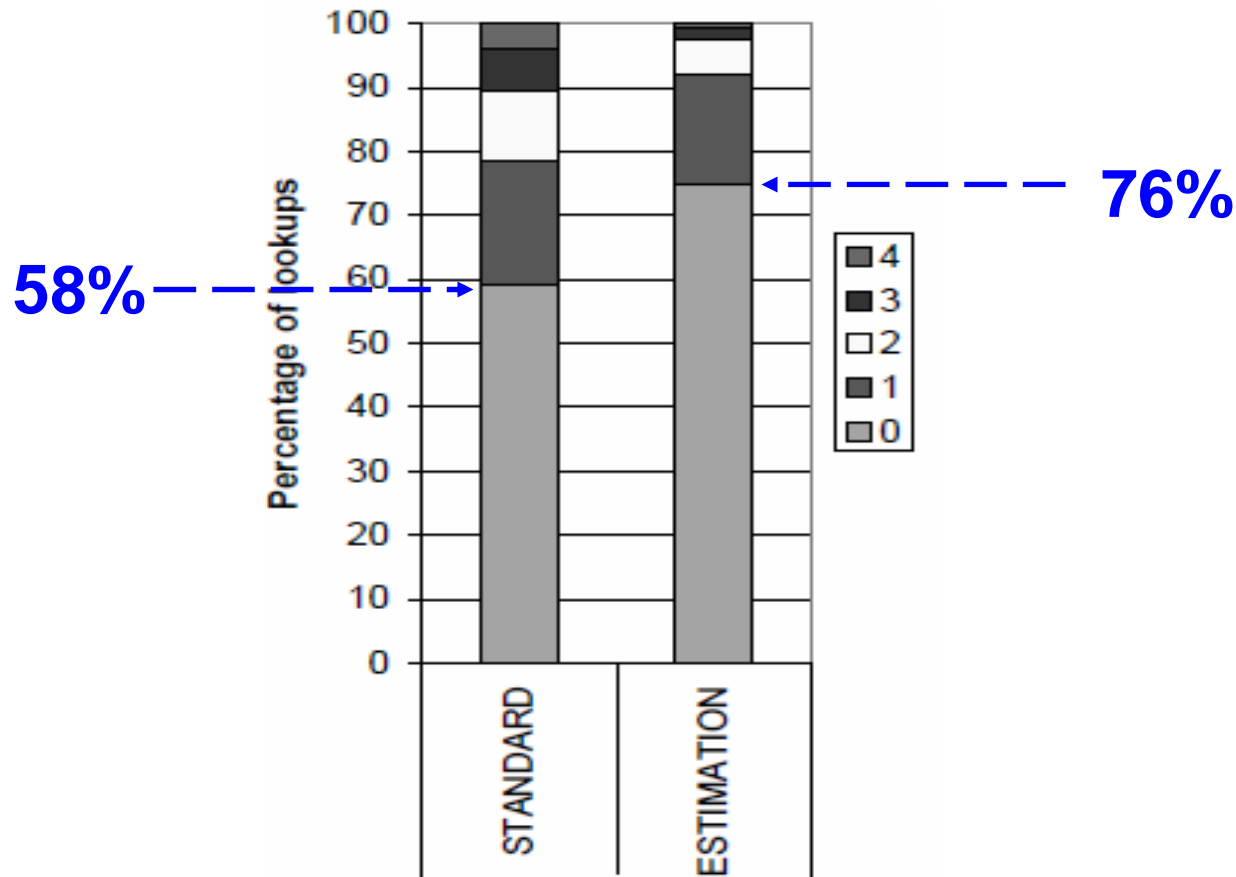
only 1.3 times long



Result – The Nearest among k Nodes

18

- Without the locality heuristic (STANDARD): 58%, 77%
- With the locality heuristic (ESTIMATION): 76%, 92%



Discussion on Pastry – Pros

19

- Pastry is scalable in terms of routing and maintaining.
 - ▣ $O(\log N)$ routing steps
 - ▣ $O(\log N)$ messages for the “join” process

- Pastry also takes into account locality.
 - ▣ Pastry finds a short routing path.
 - ▣ Chord makes no explicit effort to achieve good network locality.

Discussion on Pastry – Cons

20

- They assume that the new node X knows initially about the physically closest Pastry node A.
 - ▣ If X sends a “join” message to other nodes far from X, the locality property is not preserved.

- What is the locality heuristic?
 - ▣ There is not enough description.

- Pastry has more complexity than Chord.
 - ▣ More node states

What is PAST?

21

A large-scale peer-to-peer persistent storage utility

- PAST is layered on top of Pastry.
 - ▣ Replicas of a file are stored at nodes with nodeld matching most closely the fileld.
- PAST provides storage management capability.
 - ▣ Non-uniform storage node capacities and file sizes
 - ▣ Ensuring the availability of files by balancing the storage load
- PAST provides caching.
 - ▣ Non-uniform popularity of files
 - ▣ Minimizing client access latencies by balancing the query load

File Identifier

22

- **160-bit fileid**
 - **128 msb** are actually used **as a key** in Pastry.

- Randomly assigned when a file is stored in PAST
 - Hash (SHA-1) of name, owner public key, random salt
- Uniformly distributed in the circular Pastry Id space.
 - The number of files stored by each node is roughly balanced.
(but still improvable by storage management)

- **A file is stored on the k PAST nodes whose nodelds are numerically closest to the fileid.**
- A file can be retrieved with a fileid by Pastry routing.

Storage Management – Goals

23

- Causes of storage load imbalance
 - ▣ Statistical variation in the assignment of nodes and files
 - ▣ Non-uniform distribution of node capacities
 - ▣ Non-uniform distribution of inserted file sizes

- Bad impacts of storage load imbalance
 - ▣ Not all of the k closest nodes can accommodate a file replica
 - ▣ Low global storage utilization

- Goals
 - ▣ **Achieve high global storage utilization** by balancing the remaining free storage space among nodes
 - ▣ At the same time, **maintain the invariant of the deterministic location of k replicas**

Storage Management – Per-node Storage

24

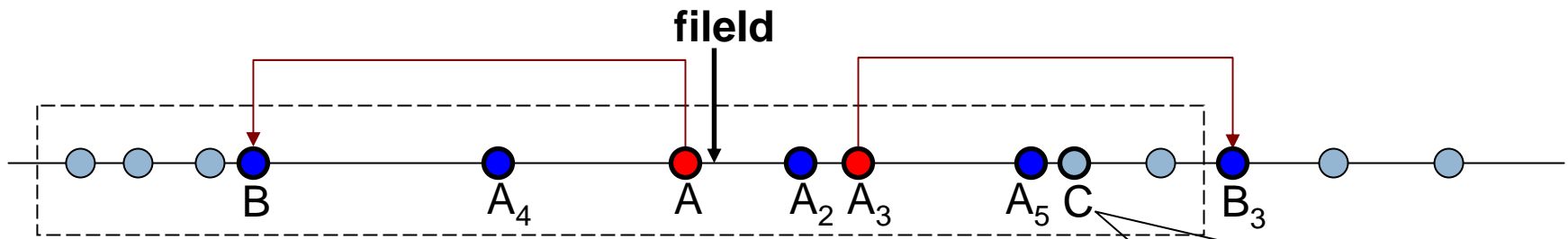
- Assumption: Storage capacities are not much different
 - ▣ Differ by no more than two orders of magnitude
- Control per-node storage capacity
 - ▣ Too much storage: split into different nodes
 - ▣ Too little: reject node join

Storage Management – Replica Diversion (1)

25

Balance the unused space among the nodes in the leaf set

- If a node A cannot accommodate a copy locally, A chooses a node B in its leaf set not among the k closest nodes.
 - ▣ A (primary node) has the pointer to B (diverted node) so that the file can be found deterministically.
 - ▣ The $(k+1)^{\text{th}}$ closest node C keeps the file table in case A fails.



● k replicas (k=5)

● Non-capable nodes among k closest nodes

□ Leaf set of A₁

File table of C

primary	diverted
A	B
A ₃	B ₃

Storage Management – Replica Diversion (2)

26

- Policy of replica diversion
 - ▣ If $S_D/F_N \leq t_{pri}$ → primary node N stores file D
 - ▣ Else → N rejects D, D is diverted to N'
 - If $S_D/F_{N'} \leq t_{div}$ → diverted node N' stores file D
 - Else → N' rejects D

- Note about this policy
 - ▣ **Avoid unnecessary diversion** when the utilization is low
 - ▣ Tend to **divert large files first** while leaving room for small files
 - ▣ $t_{pri} > t_{div}$ ensures that a diverted node has more space

Storage Management – File Diversion

27

Balance the unused space among different portions of the nodeid space in PAST

- When a file insert operation fails even in the replica diversion, the client node generates a new fileid using a different salt value.
 - ▣ NACK sent back to the client
 - ▣ Retries up to 3 times, abort after 4th failure

Maintaining Replica

28

- Node join/leave causes responsibility shift
 - ▣ PAST detects responsibility shifts when the leaf set is updated by Pastry.
 - ▣ To use network bandwidth efficiently, newly responsible node can have a pointer to an old node and copy later.
 - ▣ No node in leaf set can take over responsibility
 - Increase diverted node search space from $|L|$ to $2|L|$ nodes by contacting the node with the smallest or largest nodeId in L

Caching

29

- Basically with Pastry's locality heuristic, a client can reach the physically closest node among k nodes.
 - ▣ Not enough to handle non-uniform popularity of files

- A highly popular file demanding more than k replicas
 - ▣ As part of a lookup or insert operation, cached copies are stored along routing path between client and fileId.
 - ▣ Cache replacement policy based on GreedyDual-size algorithm
 - Weight per file: $H = 1/\text{size}(\text{file})$
 - Evict a file with the minimum H

Result – Effect of Storage Management

30

- Replica diversion and file diversion

	Without diversion	With diversion
Failure of insertion	51.1%	< 6%
Utilization	60.8%	> 94%

Dist. Name	Succeed	Fail	File diversion	Replica diversion	Util.
$l = 16$					
d_1	97.6%	2.4%	8.4%	14.8%	94.9%
d_2	97.8%	2.2%	8.0%	13.7%	94.8%
d_3	96.9%	3.1%	8.2%	17.7%	94.0%
d_4	94.5%	5.5%	10.2%	22.2%	94.1%
$l = 32$					
d_1	99.3%	0.7%	3.5%	16.1%	98.2%
d_2	99.4%	0.6%	3.3%	15.0%	98.1%
d_3	99.4%	0.6%	3.1%	18.5%	98.1%
d_4	97.9%	2.1%	4.1%	23.3%	99.3%

Table 2: Effects of varying the storage distribution and leaf set size, when $t_{pri} = 0.1$ and $t_{div} = 0.05$.

Result – Varying Threshold

31

- As t_{pri} is increased
 - ▣ Fewer files are successfully inserted
 - ▣ Higher storage utilization is achieved
 - ▣ A large file is more likely to be stored with higher threshold value.
- Varying t_{div} shows the similar results.

t_{pri}	Succeed	Fail	File divers.	Replica divers.	Util.
0.5	88.02%	11.98%	4.43%	18.80%	99.7%
0.2	96.57%	3.43%	4.41%	18.13%	99.4%
0.1	99.34%	0.66%	3.47%	16.10%	98.2%
0.05	99.73%	0.27%	2.17%	12.86%	97.4%



More failures

Higher utilization

Table 3: Insertion statistics and utilization of PAST as t_{pri} is varied and $t_{div} = 0.05$.

Result – When Diversion begins

32

- When storage utilization < 83% → Diversion negligible
 - ▣ Avoiding unnecessary replica diversion and file diversion

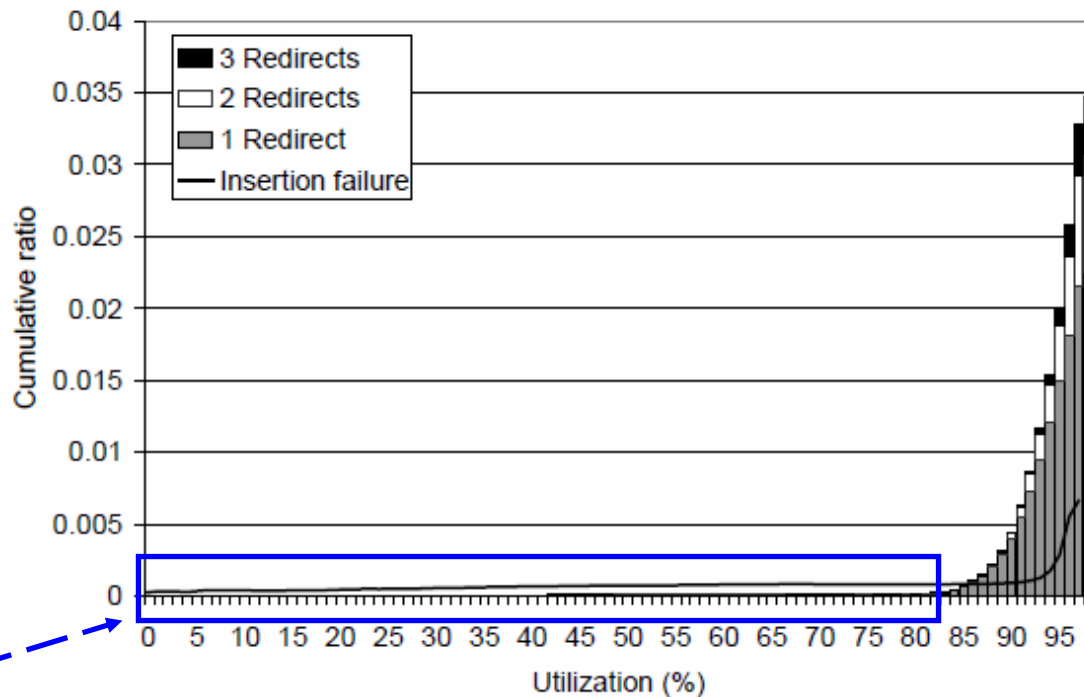


Figure 4: Ratio of file diversions and cumulative insertion failures versus storage utilization, $t_{pri} = 0.1$ and $t_{div} = 0.05$.

Result – Caching

33

- Storage utilization increases → Cache hit decreases
- GreedyDual-Size is better than LeastRecentlyUsed

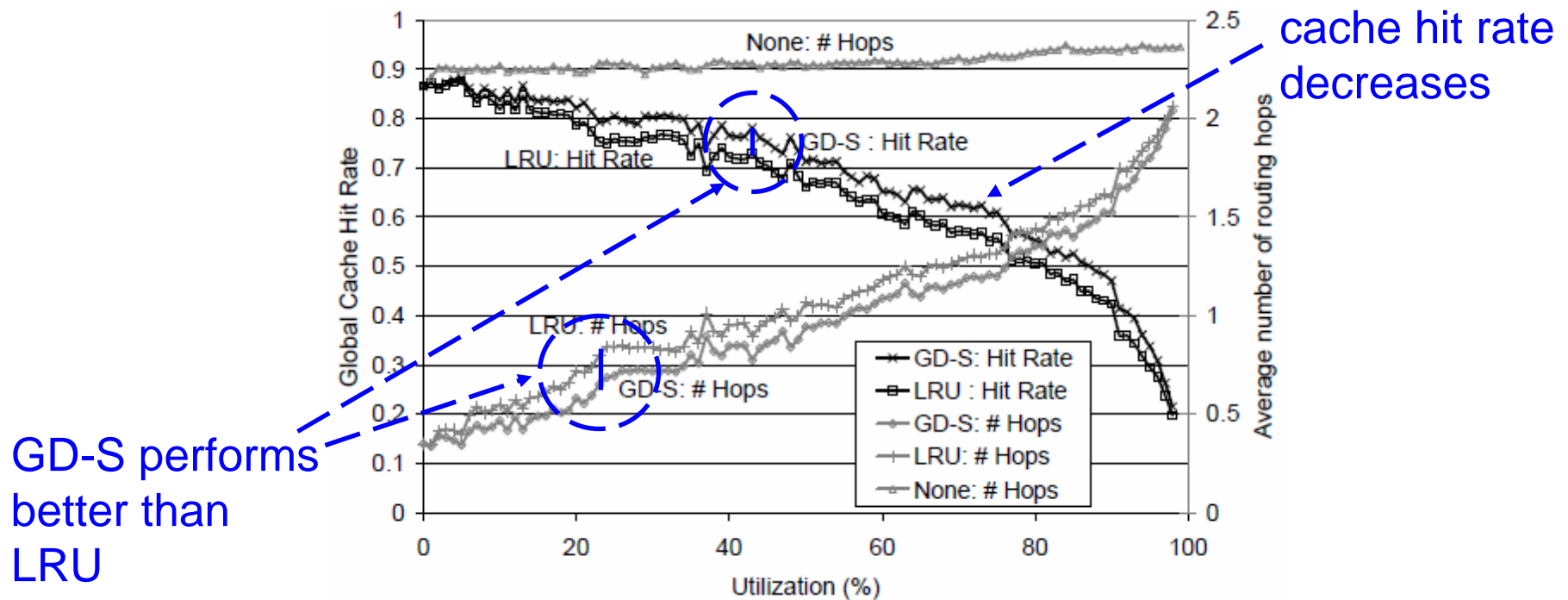


Figure 8: Global cache hit ratio and average number of message hops versus utilization using Least-Recently-Used (LRU), GreedyDual-Size (GD-S), and no caching, with $t_{pri} = 0.1$ and $t_{div} = 0.05$.

Discussion on PAST

34

□ Pros

- PAST achieves high global storage utilization.
 - With replica diversion and file diversion
 - CFS doesn't show any experimental result about global utilization.

□ Cons

- How to deal with arbitrary large files?
 - CFS can handle this by dividing a file into blocks.
- Speed for downloading a whole file from a distant node
 - CFS gets a speedup by prefetching multiple blocks at a time.
- Complexity in storage management
 - Maintaining the file table for replica diversion