Peer-to-Peer Networks
11 Past

Christian Schindelhauer
Technical Faculty
Computer-Networks and Telematics
University of Freiburg

PAST

- **PAST: A large-scale, persistent peer-to-peer storage utility**
  - by Peter Druschel (Rice University, Houston – now Max-Planck-Institut, Saarbrücken/Kaiserlautern)
  - and Antony Rowstron (Microsoft Research)
- **Literature**
  - A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", 18th ACM SOSP'01, 2001.
    - all pictures from this paper
  - P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", HotOS VIII, May 2001.

Goals of PAST

- Peer-to-Peer based Internet Storage
  - on top of Pastry
- Goals
  - File based storage
  - High availability of data
  - Persistent storage
  - Scalability
  - Efficient usage of resources

Motivation

- Multiple, diverse nodes in the Internet can be used
  - safety by different locations
- No complicated backup
  - No additional backup devices
  - No mirroring
  - No RAID or SAN systems with special hardware
- Joint use of storage
  - for sharing files
  - for publishing documents
- Overcome local storage and data safety limitations

Interface of PAST

- Create:

  `fileId = Insert(name, owner-credentials, k, file)`

  - stores a file at a user-specified number k of divers nodes within the PAST network
  - produces a 160 bit ID which identifies the file (via SHA-1)

- Lookup:

  `file = Lookup(fileId)`

  - reliably retrieves a copy of the file identified fileId

- Reclaim:

  `Reclaim(fileId, owner-credentials)`

  - reclaims the storage occupied by the k copies of the file identified by fileId

Interface of PAST

- Other operations do not exist:
  - No erase
    - to avoid complex agreement protocols
  - No write or rename
    - to avoid write conflicts
  - No group right management
    - to avoid user, group managements
  - No list files, file information, etc.
- Such operations must be provided by additional layer

Relevant Parts of Pastry

- Leafset:
  - Neighbors on the ring
- Routing Table
  - Nodes for each prefix + 1 other letter
- Neighborhood set
  - set of nodes which have small TTL

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

Interfaces of Pastry

- route(M, X):
  - route message M to node with nodeId numerically closest to X
- deliver(M):
  - deliver message M to application
- forwarding(M, X):
  - message M is being forwarded towards key X
- newLeaf(L):
  - report change in leaf set L to application

Insert Request Operation

- Compute **fileId** by hashing
  - file name
  - public key of client
  - some random numbers, called salt
- **Storage** (k x filesize)
  - is debited against client's quota
- **File certificate**
  - is produced and signed with owner's private key
  - contains fileID, SHA-1 hash of file's content, replciation factor k, the random salt, creation date, etc.

Insert Request Operation

- File and certificate are routed via Pastry
  - to node responsible for fileID
- When it arrives in one node of the k nodes close to the fileId
  - the node checks the validityof the file
  - it is duplicated to all other k-1 nodes numerically close to fileId
- When all k nodes have accepted a copy
  - Each nodes sends store receipt is send to the owner
- If something goes wrong an error message is sent back
  - and nothing stored

Lookup

- Client sends message with requested fileId into the Pastry network
- The first node storing the file answers
  - no further routing
- The node sends back the file
- Locality property of Pastry helps to send a close-by copy of a file

Reclaim

- Client‘s nodes sends reclaim certificate
  - allowing the storing nodes to check that the claim is authentificated
- Each node sends a reclaim receipt
- The client sends this recept to the retrieve the storage from the quota management

Security

- Smartcard
  - for PAST users which want to store files
  - generates and verifies all certificates
  - maintain the storage quotas
  - ensure the integrity of nodeID and fileID assignment
- Users/nodes without smartcard
  - can read and serve as storage servers
- Randomized routing
  - prevents intersection of messages
- Malicious nodes only have local influence

Storage Management

- Goals
  - Utilization of all storage
  - Storage balancing
  - Providing k file replicas
- Methods
  - Replica diversion
    - exception to storing replicas nodes in the leafset
  - File diversion
    - if the local nodes are full all replicas are stored at different locations

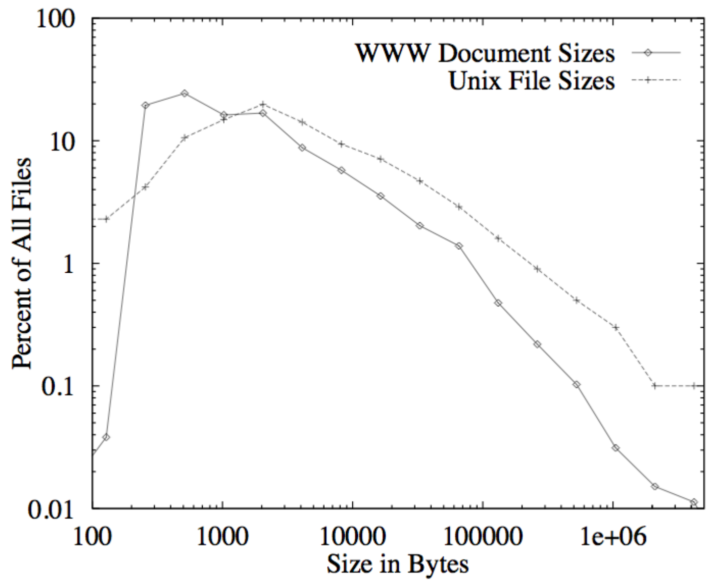
Causes of Storage Load Imbalance

- Statistical variation
  - birthday paradoxon (on a weaker scale)
- High variance of the size distribution
  - Typical heavy-tail distribution, e.g. Pareto distribution
- Different storage capacity of PAST nodes

Heavy Tail Distribution

- Discrete Pareto Distribution for x ∈ {1,2,3,…}
  - with constant factor $\zeta(\alpha) = \sum_{i=1}^{\infty} \frac{1}{i^\alpha}$
- Heavy tail
  - only for small k moments $E[X^k]$ are defined
  - Expectation is defined only if α>2
  - Variance and $E[X^2]$ only exist if α>3
  - $E[X^k]$ is defined ony if α>k+1
- Often observed:
  - Distribution of wealth, sizes of towns, frequency of molecules, ...,
  - file length, WWW documents
    - Heavy-Tailed Probability Distributions in the World Wide Web, Crovella et al. 1996

$$\mathbf{P}[X = x] = \frac{1}{\zeta(\alpha) \cdot x^\alpha}$$



Per-Node Storage

- Assumption:
  - Storage of nodes differ by at most a factor of 100
- Large scale storage
  - must be inserted as multiple PAST nodes
- Storage control:
  - if a node storage is too large it is asked to split and rejoin
  - if a node storage is too small it is rejected

Replica Diversion

- The first node close to the fileId checks whether it can store the file
  - if yes, it does and sends the store receipt
- If a node A cannot store the file, it tries replica diversion
  - A chooses a node B in its leaf set which is not among the k closest asks B to store the copy
  - If B accepts, A stores a pointer to B and sends a store receipt
- When A or B fails then the replica is inaccessible
  - failure probability is doubled

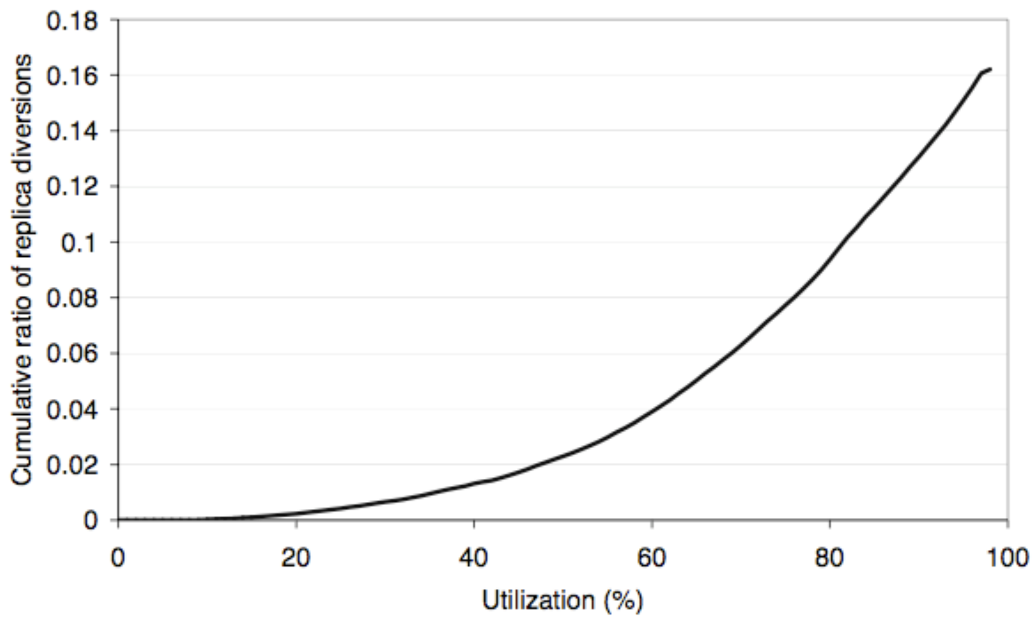


Figure 5: Cumulative ratio of replica diversions versus storage utilization, when $t_{pri} = 0.1$ and $t_{div} = 0.05$.

Policies for Replica Diversion

- Acceptance of replicas at a node
  - If (size of a file)/(remaining free space) > t then reject the file
    - for different t`s for close nodes (t_{pri}) and far nodes (t_{div}), where $t_{pri} > t_{div}$
  - discriminates large files and far storage
- Selecting a node to store a diverted replica
  - in the leaf set and
  - not in the k nodes closest to the fileId
  - do not hold a diverted replica of the same file
- Deciding when to divert a file to different part of the Pastry ring
  - If one of the k nodes does not find a proxy node
  - then it sends a reject message
  - and all nodes for the replicas discard the file

File Diversion

- **If k nodes close to the chosen fileId**
  - cannot store the file
  - nor divert the replicas locally in the leafset
- **then an error message is sent to the client**
- **The client generates a new fileId using different salt**
  - and repeats the insert operation up to 3 times
  - then the operation is aborted and a failure is reported to the application
- **Possibly the application retries with small fragments of the file**

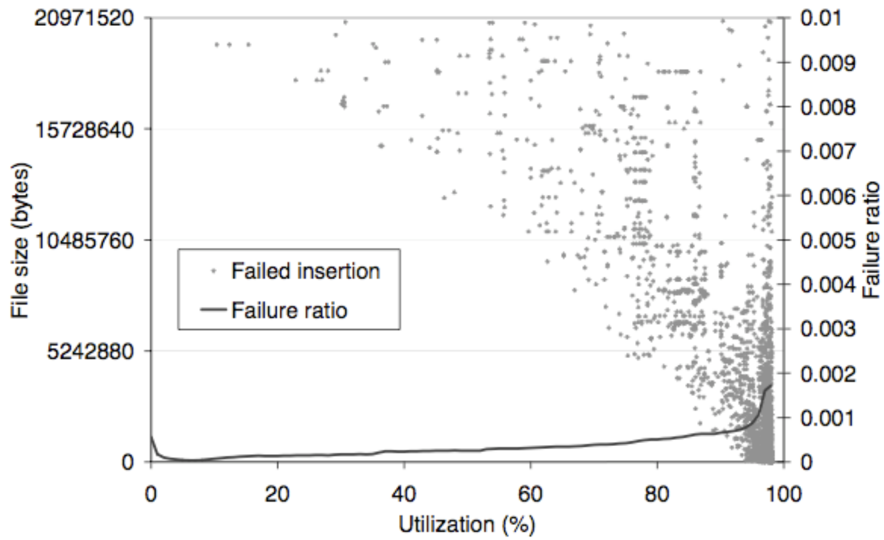


Figure 7: File insertion failures versus storage utilization for the filesystem workload, when $t_{pri} = 0.1$, $t_{div} = 0.05$.

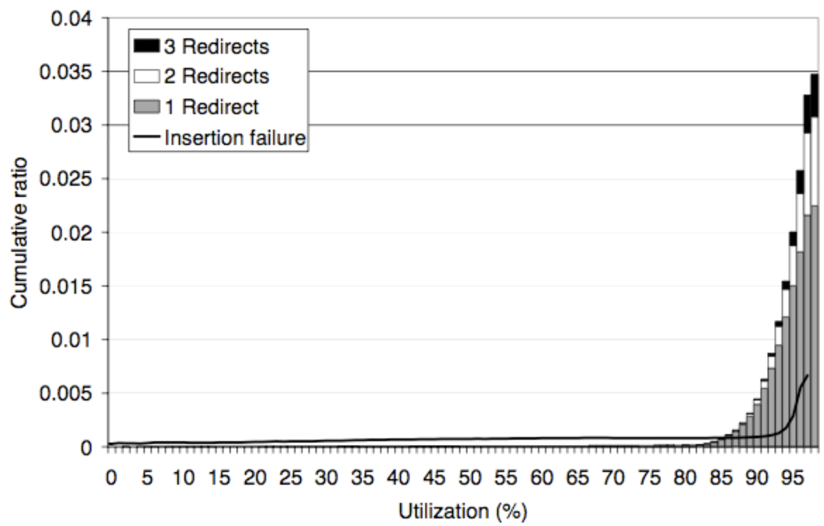


Figure 4: Ratio of file diversions and cumulative insertion failures versus storage utilization, $t_{pri} = 0.1$ and $t_{div} = 0.05$.

Maintaining Replicas

- Pastry protocols checks leaf set periodically
- Node failure has been recognized
    - if a node is unresponsive for some certain time
  - Pastry triggers adjustment of the leaf set
    - PAST redistributes replicas
  - if the new neighbor is too full, then other nodes in the nodes will be uses via replica diversion
- When a new node arrives
  - files are not moved, but pointers adjusted (replica diversion)
  - because of ratio of storage to bandwidth

File Encoding

- k replicas is not the best redundancy strategy
- Using a Reed-Solomon encoding
  - with m additional check sum blocks to n original data blocks
  - reduces the storage overhead to (m+n)/n times the file size
    - if all m+n shares are distributed over different nodes
  - possibly speeds upt the access spee
- PAST
  - does NOT use any such encoding techniques

Caching

- Goal:
  - Minimize fetch distance
  - Maximize query throughput
  - Balance the query load
- Replicas provide these features
  - Highly popular files may demand many more replicas
    - this is provided by cache management
- PAST nodes use „unused“ portion to cache files
  - cached copies can be erased at any time
    - e.g. for storing primary of redirected replicas
- When a file is routed through a node during lookup or insert it is inserted into the local cache
- Cache replacement policy: GreedyDual-Size
  - considers aging, file size and costs of a file

Experimental Results Caching

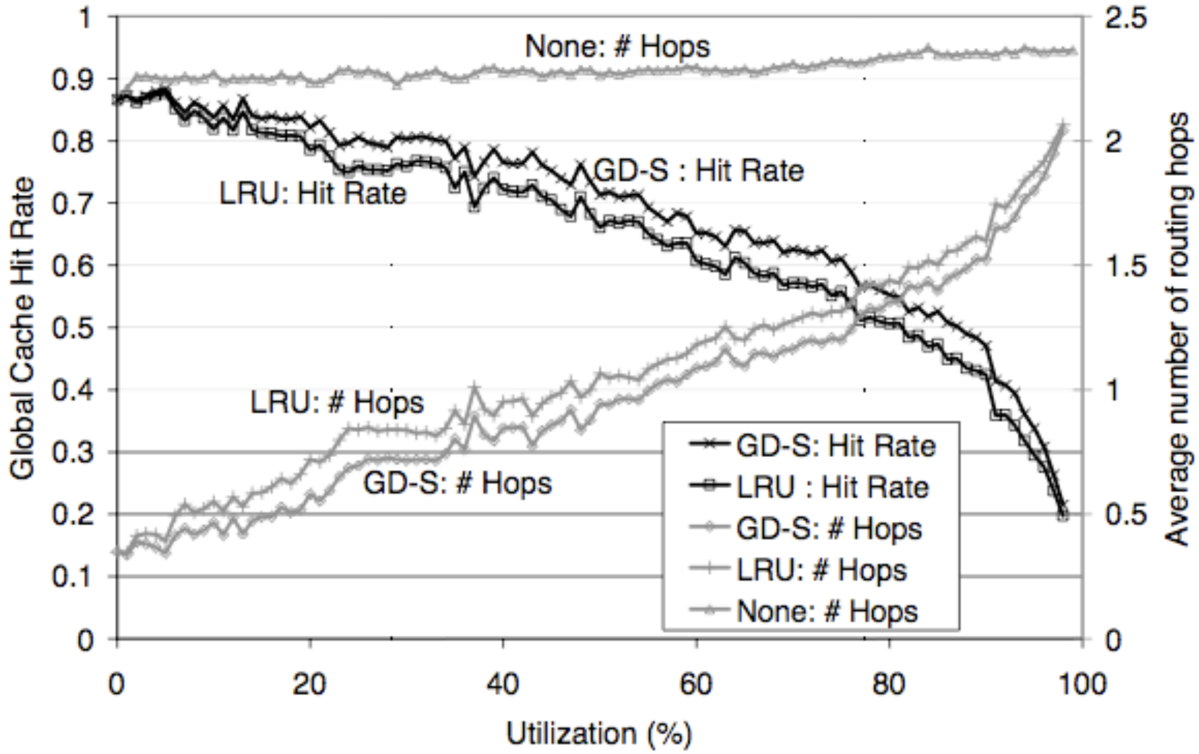


Figure 8: Global cache hit ratio and average number of message hops versus utilization using Least-Recently-Used (LRU), GreedyDual-Size (GD-S), and no caching, with $t_{pri} = 0.1$ and $t_{div} = 0.05$.

Summary

- PAST provides a distributed storage system
  - which allows full storage usage and locality features
- Storage management
  - based ond Smartcard system
    - provides a hardware restriction
  - utilization moderately increases failure rates and time behavior

Peer-to-Peer Networks

11 Past

Christian Schindelhauer
Technical Faculty
Computer-Networks and Telematics
University of Freiburg