HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Winter semester 2014

Jun.-Prof. Dr.-Ing. Kalman Graffi
Heinrich Heine University Düsseldorf

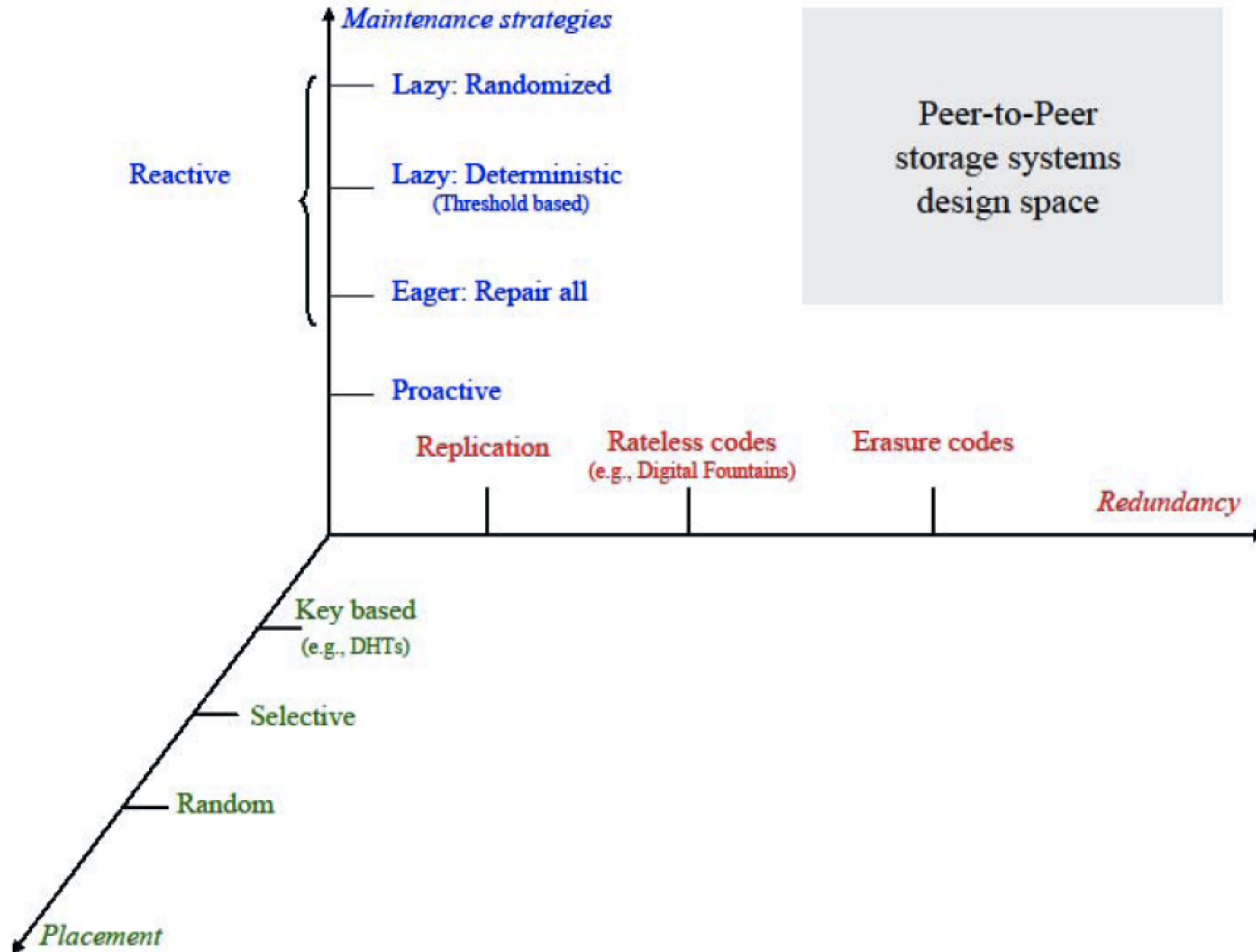
Peer-to-Peer Systems – Chapter 10

P2P Storage Systems

- Data Replication Schemes
- Cooperative File System

Design Space of P2P Storage

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF



Slide based on Anwitaman Datta, "Peer-to-Peer Storage Systems: Crowdsourcing the storage cloud", ICDCN '10

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

P2P-based Storage Systems
– Redundancy Maintenance

This slide set is based on the lecture “Peer-to-Peer Networks" of Prof. Dr. Jussi Kangasharju at University of Helsinki

Replication

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Motivation

- If file (or block) is stored only on one peer and that peer is offline, data is not available
- Replicating content to multiple peers significantly increases content availability

Benefits

- High availability and reliability
- But only probabilistic guarantees

Con:

- How to coordinate lots of replicas?
  - Especially important if content can change (or can be deleted)
- Good availability in unreliable networks requires high degree of replication
  - “Wastes” storage space

Redundancy through full copy replication

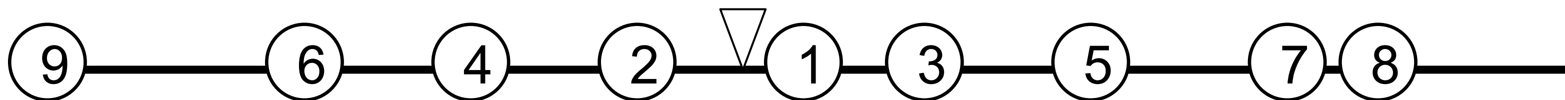
Traditional replication:

- Several copies of same content

Example PAST (extension to Pastry):

Nodes [] replicaSet (key → k, int → max rank)

- Returns an ordered set of peers of magnitude (max rank) on which replicas of the object with key k can be stored
- The nodes which become roots for the key k when the local node fails



Redundancy through full copy replication

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Traditional replication:

- Several copies of same content

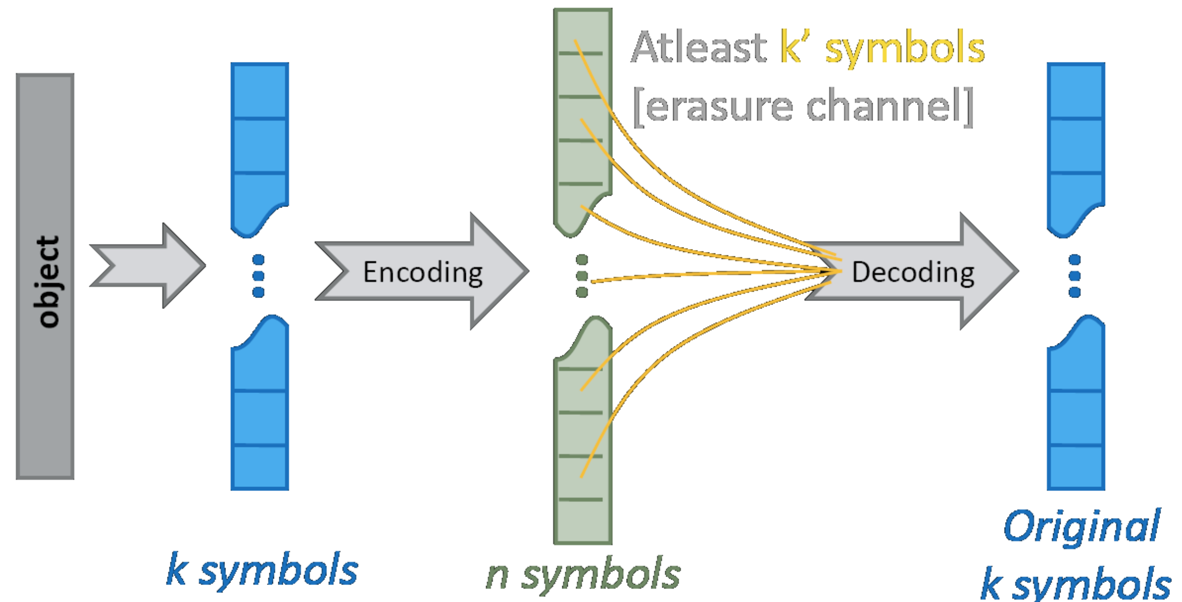
Other approaches:

- Using several object IDs
  - ID1 = hash(Object)
  - ID2 = hash(ID1)
  - IDi+1 = hash(IDi)

Erasure Codes

Erasure codes

- Split content in blocks (symbols) (e.g. 50)
- Create redundancy in blocks
  - → e.g. 200 blocks
- For restoring content
  - 50 out of 200 blocks needed



Slide based on Anwitaman Datta, "Peer -to-Peer Storage Systems: Crowdsourcing the Storage Cloud", ICDCN'10

Rateless Erasure Codes

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

A class of erasure codes with the property that

- potentially limitless sequence of encoding symbols can be generated from a given set of source symbols
- such that the original source symbols can be recovered
  - from any subset of the encoding symbols of size equal to or only slightly larger than the number of source symbols

Examples

- Digital fountains, Online codes, …

Implications for P2P storage

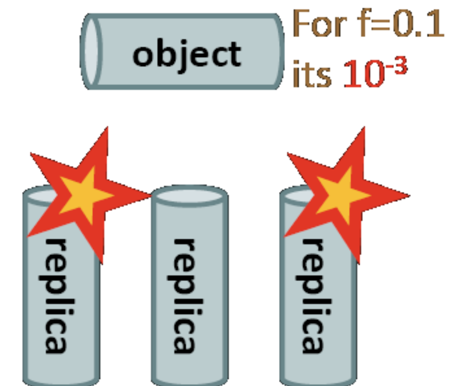
- From maintenance perspective: a hybrid of replication/traditional erasures

Slide based on Anwitaman Datta, "Peer-to-Peer Storage Systems: Crowdsourcing the Storage Cloud", ICDCN'10

Erasure Codes: Static Resilience

Replicated r times

- Faults that can be tolerated: r - 1
- Probability of failure: f^r
- Storage efficiency: 1/r
- Access: Find any one good replica

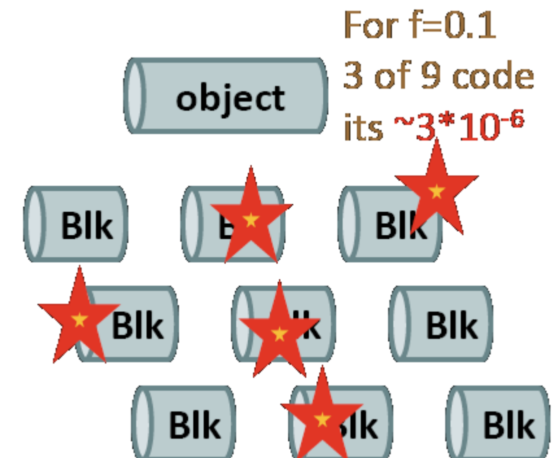


Erasure coded (k of n)

- Faults that can be tolerated: n-k
- Probability of failure:

$$\sum_{j=1}^{k} \binom{n}{n-k+j} f^{n-k+j}(1-f)^{k-j}$$

- Storage efficiency: k/n
- Access: Find k good blocks



Assume: Peer failure is i.i.d. with failure probability f

Slide based on Anwitaman Datta, “Peer -to-Peer Storage Systems: Crowdsourcing the Storage Cloud“, ICDCN'10

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Example: Static Resilience

Churn rate, f=0.25

Replication

- r = 5
    - → Availability ~ 0.73
    - Too low
- r = 24
    - → Availability > 0.999
    - Too much overhead!

Erasure code

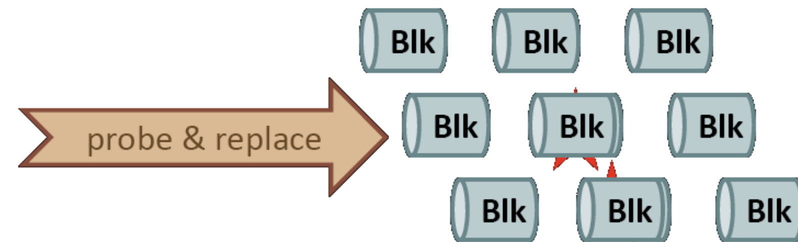
- N=517, k=100
    - → Availability ~ 0.999
    - Overhead = 5.17

Slide based on Anwitaman Datta, "Peer -to-Peer Storage Systems: Crowdsourcing the Storage Cloud", ICDCN'10

Maintaining redundancy

Why to maintain redundancy?

- Churn

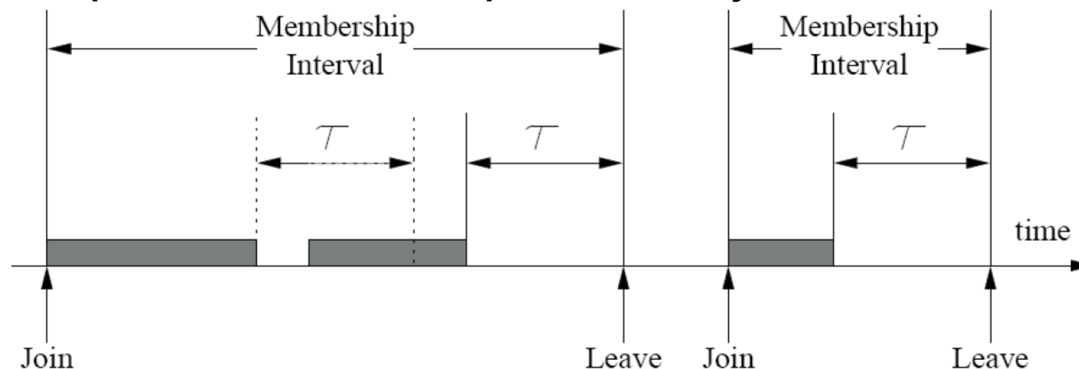


Naïve solution (Eager maintenance)

- Probe (e.g., periodically) and replace "lost" redundancy

How do we know whether a peer left permanently?

- Heuristic time out T might help
- Avoids replications, when peer is only offline for a short time



Lazy Maintenance

Basic idea

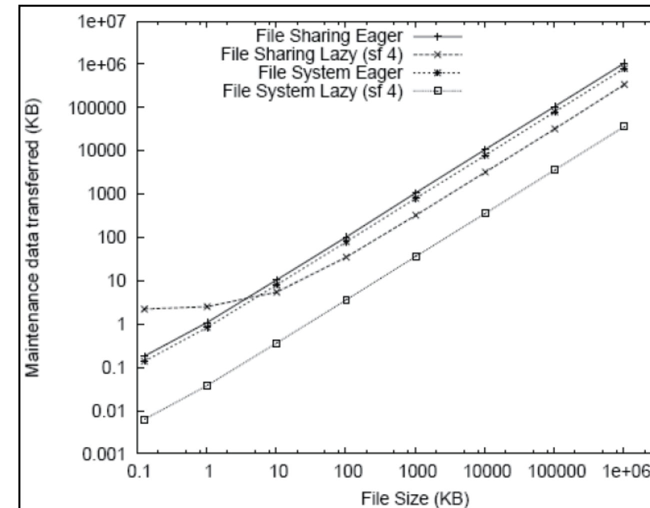
- After probing for all blocks,
- if the number of available blocks (a) is below a repair threshold (th)
- then replace all missing blocks (deterministic lazy repair)

Approach

- If originally n blocks were being stored, then n-a new blocks regenerated
- Th is a design parameter
- If (n,k) erasure coding is being used, k < th

Evaluation

- + Saves bandwidth, however peaky
- - System complexity
- - Meta−info overheads
  - Start with extra redundancy
  - Otherwise system vulnerable



Total Recall: System Support for Automated Availability Management
R. Bhagwan, K. Tati, Y. Cheng, Stefan Savage, and G. Voelker. NSDI 2004

Randomized Lazy Maintenance

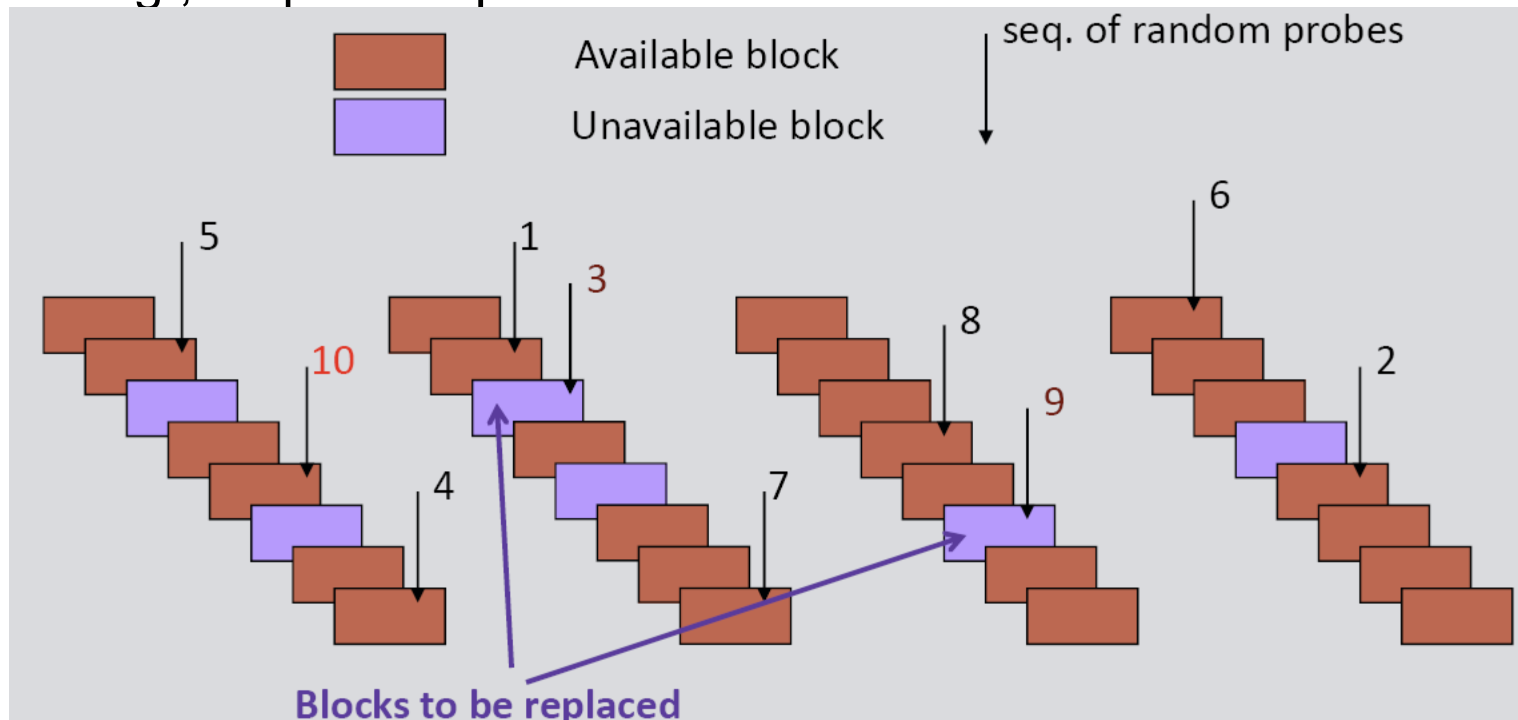
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Randomly probe the redundant nodes

- till a threshold (tr) live blocks are found (or all nodes probed)
- E.g. Probe randomly till 8 online blocks are found

Replenish the blocks detected to be missing

- E.g., 10 probes: probes 3 and 9 detect 2 out of 6 unavailable



Slide based on Anwitaman Datta, "Peer -to-Peer Storage Systems: Crowdsourcing the Storage Cloud", ICDCN'10

Randomized Lazy Maintenance

Number of replacements

- Generally less than the total number of unavailable blocks

A continuous but slower process

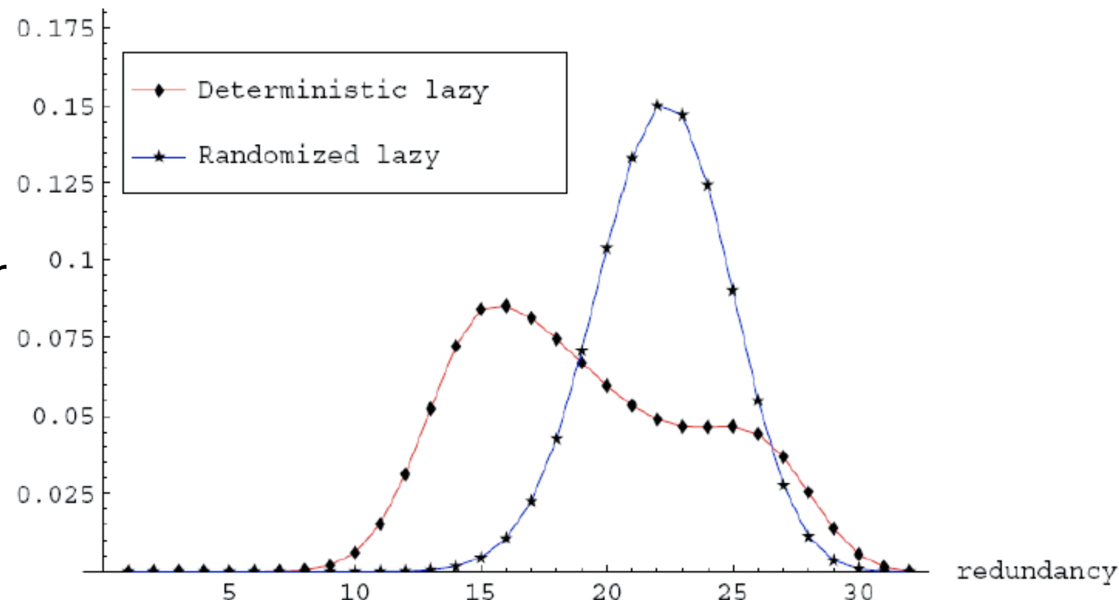
- thus smoother bandwidth usage

System is in a healthier shape

- Inherently adaptive

System complexity

- is (again) higher
- than the naïve solution



Internet-scale storage systems under churn - A steady state analysis, A. Datta, K. Aberer, IEEE P2P 2006

Slide based on Anwitaman Datta, "Peer -to-Peer Storage Systems: Crowdsourcing the Storage Cloud", ICDCN'10

Proactive Replication Strategy

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

A radically different idea

Assumption:

- Total bandwidth and storage usage are not such a big deal
- But the bandwidth used at any time instant is capped to a max-level

Approach:

- Continuously create extra redundancy (till some N_max)

Proactive replication for data durability.
E. Sit, A. Haeberlen, F. Dabek, B. Chun, H. Weatherspoon, R. Morris, M. Frans Kaashoek and J. Kubiatowicz, IPTPS 2006

Slide based on Anwitaman Datta, "Peer -to-Peer Storage Systems: Crowdsourcing the Storage Cloud", ICDCN'10

Garbage collection

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

What happens when nodes come back online?

- Duplicates and its implications
  - Replication vs. Erasure codes vs. Rateless codes

Garbage collection decisions

- To keep duplicates or not to keep duplicates
  - Affects system complexity
  - Keep track of duplicates and/or detect duplicates to remove
- For coded fragments, limit the degree of diversity to keep (N_max)

Tricky question:

- What happens with updates? Changes of the content?
- How to find duplicates and replicates?

Redundancy Maintenance and Garbage Collection Strategies in Peer-to-Peer Storage Systems
Liu Xin, Anwitaman Datta, in SSS 2009

P2P storage system simulator: http://code.google.com/p/p2p3s/

Slide based on Anwitaman Datta, "Peer-to-Peer Storage Systems: Crowdsourcing the Storage Cloud", ICDCN'10

Peer-to-Peer Systems

P2P-based Storage Systems
– Cooperative File System (CFS)

This slide set is based on the lecture “Peer-to-Peer Networks" of Prof. Dr. Jussi Kangasharju at University of Helsinki

CFS – Cooperative File System

History

- Developed at MIT, by same people as Chord
- CFS based on the Chord DHT
- Read-only system, only 1 publisher

CFS stores blocks instead of whole files

- Part of CFS is a generic block storage layer

Features

- Load balancing
- Performance equal to FTP
- Efficiency and fast recovery from failures

CFS Properties

Scalability (comes from Chord)

Availability

- In absence of catastrophic failures, of course…
- Not-adaptive replication strategy

Balanced number of files

- Storage load balanced according to peers' capabilities
- See AH-Chord: traffic load is not balanced in Chord

Persistence

- Data will be stored for as long as agreed

Quotas

- Possibility of per-user quotas

Efficiency

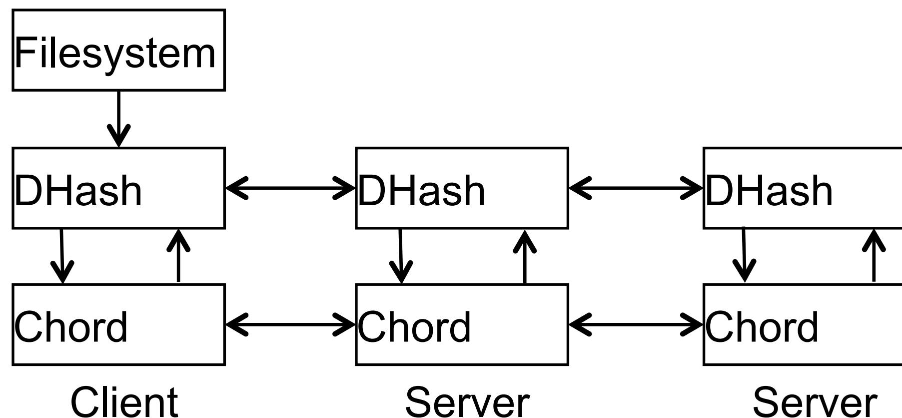
- As fast as common FTP in wide area

CFS: Layers, Clients, and Servers

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Components

- FS: provide filesystem API, interpret blocks as files
- DHash: Provides block storage
- Chord: DHT layer, slightly modified
  - Instead of 1 successor: r successors
  - Node-to-node latency is measured and communicated
- Clients access files, servers just provide storage



DHash Layer

DHash stores blocks as opposed to whole files

- Better load balancing
- More network query traffic, but not really significant

Each block replicated k times, with r ≥ k

Two kinds of blocks:

- Content block, addressed by hash of contents
- Signed blocks (= root blocks), addressed by public key
  - Signed block is the root of the filesystem
  - One filesystem per publisher

Blocks are cached in network

- Most of caching near the responsible node
- Blocks in local cache replaced according to least-recently-used
- Consistency not a problem, blocks addressed by content
  - Root blocks different, may get old (but consistent) data

DHash API

DHash provides following API to clients:

- Put_h(block) : Store block
- Put_s(block, PubKey) : Store or update signed block
- Get(key) : Fetch block associated with key

Filesystem starts with root (= signed) block

- Block under publisher's public key and signed with private key
  - Clients: block is read-only
  - Publisher can modify filesystem by inserting a new root block
- Root block has pointers to other blocks
  - Either piece of a file or filesystem metadata (e.g., directory)

Data stored for an agreed-upon finite interval

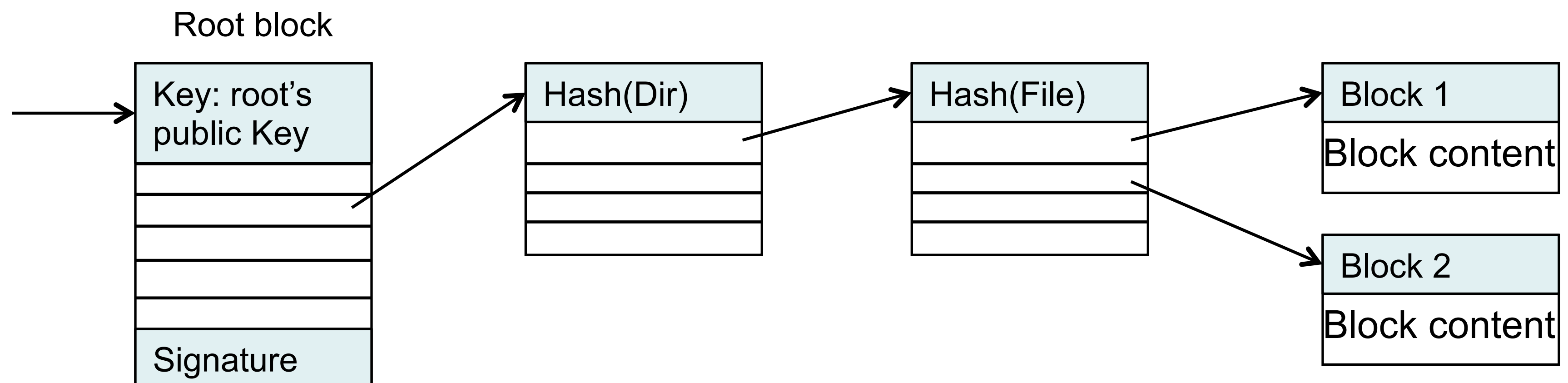
CFS Filesystem Structure

Root block identified by publisher's public key

- Each publisher has its own filesystem
- Different publishers are on separate filesystems

Other blocks can be metadata or pieces of file

- Metadata: contains links to other keys
- Piece of file: other blocks identified based on hash of contents



Operations in CFS

First write

- Create data elements to store
- Sign root document
- Publish data elements

CFS will store any block under its content hash

- Highly unlikely to find two blocks with same SHA-1 hash
- No explicit protection for content blocks needed

Only publisher can change data in filesystem

- All links in CFS originate in root document
- Root block is signed by publisher
  - Validity can be checked by storage peer and users
  - Publisher must keep private key secret

No explicit delete operation

- Data stored only for agreed-upon period
- Publisher must refresh periodically if persistence is needed

Peer-to-Peer Systems

P2P-based Storage Systems

– Ivy

This slide set is based on the lecture “Peer-to-Peer Networks" of Prof. Dr. Jussi Kangasharju at University of Helsinki

Ivy

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

History

- Ivy developed at MIT
- Based on Chord

Provides NFS-like semantics

- At least for fully connected networks
- Any user can modify any file
- Ivy handles everything through logs
- Ivy presents a conventional filesystem interface

Problems for Distributed Read/Write

Multiple distributed writers

- make it difficult to maintain consistent metadata

Unreliable participants make locking unattractive

- Locking could help maintain consistency

Participants cannot be trusted

- Machines may be compromised
- Need to be able to undo

Distributed filesystem may become partitioned

- System must remain operational during partitions
- Help applications repair conflicting updates made during partitions

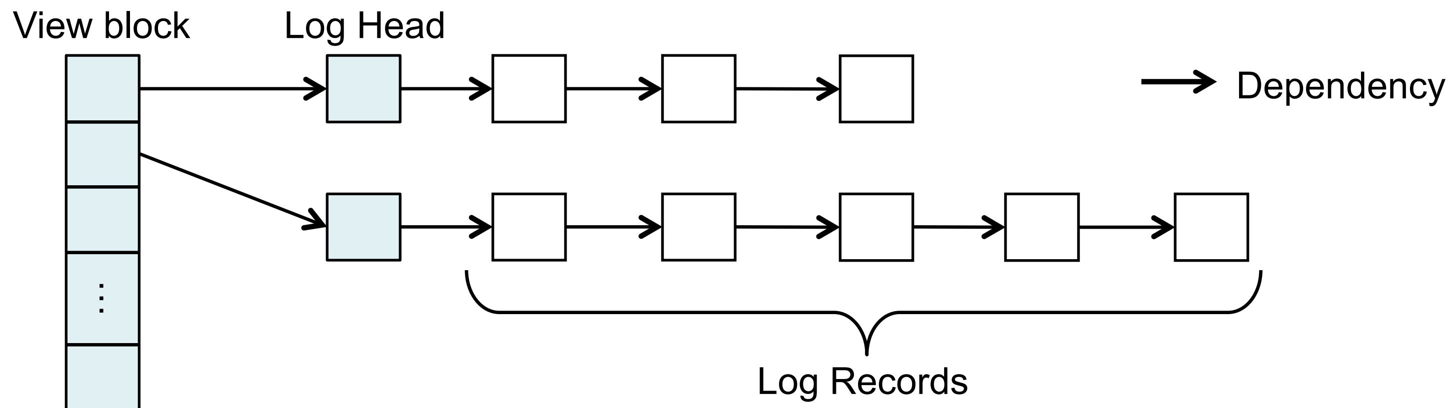
Solution Approach: Logs

Each participant maintains log of its changes

- Log: Command for modification
- Logs maintain filesystem data and metadata
- Participant has private snapshot of logs of others
- Participant writes to its own log, reads all others
  - Log-head points to most recent log record

Logs stored in DHash (see under CFS for details)

- Version vectors impose order on log records from multiple logs



Ivy: Views and Snapshots

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Views:

- Each user writing to a filesystem has its own log
- Participants agree on a view
  - View is a set of logs that comprise the filesystem
- View block is immutable (“root”)
- View block has log heads for all participants

Snapshots

- Private snapshots avoid traversing whole filesystem
  - Contents of snapshots mostly the same for all participants
  - DHash will store them only once
- Snapshot contains the entire state
- To create snapshot, node:
  - Gets all logs recent than current snapshot
  - Write new snapshot
- New user must either build from scratch or take a trusted snapshot

Combining Logs

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

How to determine order of modifications from logs?

- Order should obey causality
- All participants should agree on the order

Each new log record (= modification) has

- Sequence number: increasing (managed locally)
- Version vector: summarizes knowledge about log
  - Has sequence numbers from other logs in view

Log records ordered by comparing version vectors

- Vectors u and v comparable if $u < v$, $v < u$, or $v = u$
- Otherwise concurrent

Simultaneous operations result in equal or concurrent vectors

- Ordered by public keys of participants
- May need special actions to return to consistency
  - In case of overlapping modifications

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

P2P-based Storage Systems – OceanStore

This slide set is based on the lecture “Peer-to-Peer Networks" of Prof. Dr. Jussi Kangasharju at University of Helsinki

OceanStore

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

History:

- OceanStore developed at UC Berkeley
- Runs on Tapestry DHT
- Supports object modification

Vision of ubiquitous computing:

- Intelligent devices, transparently in the environment
- Highly dynamic, untrusted environment

Question: Where does persistent information reside?

- OceanStore aims to be the answer
- OceanStore’s target in 2003/2004:
  - 1010 users, each with 10000 files, i.e., 10M files total
- See Cloud storage today

OceanStore: Basics and Goals

Users pay for the storage service

- Several companies can provide services together

Aiming to support:

1. Untrusted infrastructure
    - Everything is encrypted, infrastructure unreliable
    - However, assume that “most servers are working correctly most of the time”
    - One class of servers trusted to follow protocol
        - But not trusted with data
2. Nomadic data
    - Anytime, anywhere
    - Introspection used to tune system at run-time

OceanStore System Overview

Objects:

- Each object has globally unique identifier (GUID)
- Objects replicated and migrated on the fly

Replicas located in two ways

- Probabilistic, fast algorithm tried first
- Slower, but deterministic algorithm used if first one fails

Objects exist in two forms, active and archival

- Active is the latest version with handle for updates
- Archival is a permanent, read-only version
  - Archive versions encoded with erasure codes with lot of redundancy
  - Only a global disaster can make data disappear

Naming and Access Control

Object naming

- Self-certifying object names
  - Hash of the object's owners key and human readable name
- Allows directories
- System has no root, but user can select her own root(s)

Access control

- Restricting readers
  - All data is encrypted, can restrict readers by not giving key
  - Revoke read permission by re-encrypting everything
- Restricting writers
  - All writes must be signed and compared against Access Control List (ACL)

Two Mechanisms for Locating Objects

Probabilistic algorithm

- Frequently accessed objects likely to be nearby and easily found
- This algorithm finds them fast
- Uses attenuated multi-level Bloom filters
    - See next slides

Deterministic algorithm

- OceanStore based on Tapestry
    - Like the p2p overlay Pastry
- Deterministic routing is Tapestry’s routing
- Guaranteed to find the object

Excursion: Bloom Filters

Bloom filters are

- *"a space-efficient probabilistic data structure that is used to test whether or not an element is a member of a set"*
- Test if element e is part of a set:
  - Answer NO:
    - Answer is guaranteed
    - False negatives are NOT possible
  - Answer MAYBE:
    - Element might be in set
    - False positives are possible

Components of an bloom filter:

- Array of k bits
- Also need m different hash functions,
  - Each hash function maps input to an array index ({0,k-1}): bit at index set then to 1

To insert

- Calculate all m hash functions (→ m indices) and set bits at indices to 1

To check

- Calculate all m hash functions
  - If all bits are 1, key is "probably" in the set
  - If any bit is 0, then it is definitely not in

Excursion: Bloom Filters

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

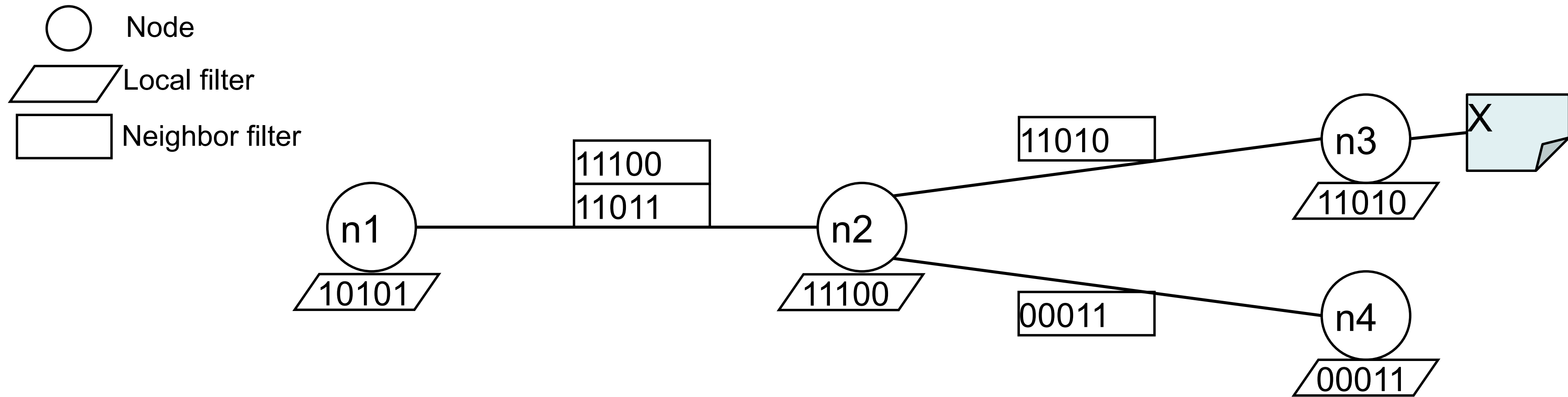
To insert or check for an item Bloom filters take O(m) time

- Fixed constant! Independent of number of entries
- No other data structure allows for this (hash table is close)

Attenuated Bloom filter of depth D: An array of D normal Bloom filters

- 0-th level filter is for locally stored objects at current node
- The i-th Bloom filter is the union of all Bloom filters at distance i
  - Through any path from current node
- Attenuated Bloom filter for each network edge
  - Gives information “What is in that direction”
- Queries routed on the edge where the distance to object is shortest
  - „Shortest“: low level of matching Bloom filter indicates promising direction

Excursion: Probabilistic Query Process - Example



Node n1 wants to find object X

- X hashes to bits 0, 1, 3 → Set bits at index 0,1,3 to 1

Node n1 does not have 0, 1, and 3 in local filter

- Neighbor filter for n2 has them, forward query to n2

Node n2 does not have them in local filter

- Filter for neighbor n3 has them, forward to n3

Node n3 has object

Update Model

Update model in OceanStore based on conflict resolution

Update semantics

- Each update has a list of predicates with associated actions
- Predicates evaluated in order
- Actions for first true predicate are atomically applied (commit)
- If no predicates are true, action aborts
- Update is logged in both cases

List of predicates is short

- Untrusted environment limits what predicates can do
- Available predicates:
  - Compare-version (metadata comparison, easy)
  - Compare-size (same as above)
  - Compare-block (easy if encryption is position-dependent block cipher)
  - Search (possible to search ciphertext, get boolean result)

Available Operations

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Four operations available
- Replace-block
- Insert-block
- Delete-block
- Append

In case of position-dependent cipher
- I.e. symmetric encryption key depends on storage key
- Replace-block and Append are easy operations

For Insert-block and Delete-block
- Two kinds of blocks: Index and data blocks
- Index blocks can contain pointers to other blocks
- To insert a block, we replace old block with an index block which points to old block and new block
  - Actual blocks are appended to object
- May be susceptible to traffic analysis

Serializing Updates

Replicas divided into two tiers

- Primary tier
  - Is trusted to follow protocol
  - Peers cooperate in a Byzantine agreement protocol
- Secondary tier is everyone else
  - Communicates with primary tier and within the second tier via epidemic algorithms

Reason for two tiers:

- Fault-tolerant protocols possible with only a small number of replicas, protocols communication-intensive
- Primary tier is well-connected and small

Byzantine Generals Problem

Scenario:

- Several divisions of the Byzantine army surround an enemy city.
- Each division is commanded by a general.

Communication

- The generals communicate only through messenger
  - Need to decide on a common plan after observing the enemy
- Some of the generals may be traitors
  - Traitors can send false messages

Required: An algorithm to guarantee that

- All loyal generals decide upon the same plan of action, irrespective of what the traitors do.
- A small number of traitors cannot cause the loyal generals to adopt a bad plan.

Solution:

- If no more than m generals out of n = 3m + 1 are traitors
- Loyal generals can agree on plan and follow the orders

Updating Data in Two Tiers

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Node in secondary tier initiates an update

- Sends update to one node in primary tier
- Sends also some replicas to random nodes in secondary tier

Update propagation

- In primary tier: Byzantine agreement is performed
  - Object eventually accepted (or rejected)
- In seconday tier: replicas are forwarded randomly (epidemically)

Update flush

- In case of positive result of Byzantine agreement:
  - Update is sent over multicast tree to all secondary replicas

Deep Archival Storage

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Archival mechanism uses erasure codes

- Reed-Solomon, Tornado, etc.

Generate redundant data fragments

- Created by the primary tier

If there are enough fragments and they are spread widely, then it is likely that we can retrieve the data

Archival copies are created when objects are changed

- Every version is archived
- Can be tuned to be done less frequently