Distributed Data Management

Christoph Lofi
Institut für Informationssysteme
Technische Universität Braunschweig
http://www.ifis.cs.tu-bs.de
9.0 Durability

9.1 Basic Chord Durability
9.2 Load Balancing
9.3 Power of Two Choices
9.4 Virtual Servers
9.5 LOCKSS & OceanStore
9.0 Basic Chord

- Remember the Chord DHT
  - **Hash Function** for hashing data and nodes alike
  - Each node is **responsible** for address arc between itself and the previous node

![Chord Ring Diagram]

- successor(6) = 7
- successor(7) = 1
- successor(1) = 6

Example key space: 0...7

- Identifier
- Node
- Data Item w. id

Distributed Data Management – Christoph Lofi – IfIS – TU Braunschweig
9.0 Basic Chord

- A new node **takes over some responsibility** from an older nodes
  - i.e. key-value pairs are **moved** to the new node
- Each node “**knows**” some other nodes
  - **Finger table** with increasingly distant nodes for $O(\log(n))$ routing
    - Finger distance based on address space
  - **Successor list** of the next $k$ nodes in ring for supporting stabilization
    - Independent from address space distance
Stabilize function continuously fixes broken finger table and successor list entries
  • Links to left / unreachable / failed nodes will be repaired
  • DHT routing will be resilient to failures

But: Basic Chord does not offer any data durability
  • Direct Storage:
    – Stored data and tuples are lost when a node is fails!
  • Indirect Storage
    – Uses soft states to ensure timely updates of indirect links
    – Data is lost if data providing node fails!

This lecture: How can we introduce data durability to Chord?
• More issues with basic Chord
  – Hash function **evenly distributes** keys and nodes across the address space
    • Basic idea of hashing: even load distribution to the buckets
  – But: often, this will not result in a load balanced system
    • User queries are usually not evenly distributed
      – “**Hot topics**” and “**Long Tail**”; i.e. data everybody wants and data nearly nobody wants
    • Also, even using a good hash function will not result in equal load distribution for nodes
      – Balancing necessary

• Also this lecture: **Load Balancing for DHTs**
For archiving durability in Chord, replication is needed

- **Simple Replication Strategies**
  - Just keep multiple copies
  - Create new copies if a copy is lost

- **Load Balancing Replication**
  - Keep multiple copies
  - Keep more copies of popular or high-in-demand data
9.1 Basic Chord Durability

- Multiple Copies using **Successor List**
  - Store data at responsible node
    - Additionally, *replicate* data to the $k$ next other nodes
  - After a node fails, **stabilize** will repair routing
    - After routing is repaired, *replicate* to the next successor/s until data is again replicated to $k$ additional nodes
Advantages
- After a node failure, its successor has the data already stored
  • System function is not interrupted

Disadvantages
- Node stores $k$ intervals
  • More data load
  • Data localization more fuzzy
- After breakdown of a node
  • Find new successor
  • Replicate data to next successor
    - Message overhead during repair
- Stabilize-function has to check every successor-list
  • Find inconsistent links
    - More message overhead
### 9.1 Basic Chord Durability

- **Multiples nodes per interval**
  - Responsibility of an address arc is fully shared by at least $k$ nodes
  - New nodes arriving will be assigned to an arc
    - New node obtains a copy of all arc data
    - Responsibility is only split if $k$ is significantly exceeded
      - e.g. $2k$
      - New arc segment will have $k$ responsible nodes
  - **New link structure**: links to other nodes in same interval
    - New nodes are announced to all other nodes in interval
  - Also possible: pass new node on to the next interval if already full
9.1 Basic Chord Durability

- **Data Insertion**
  - Replicate data to all other nodes in arc

- **Failure**
  - No copy of data needed
  - Data are already stored within same interval
  - If arc is critically low, borrow nodes from neighbor arcs

- **Use stabilization procedure to correct fingers**
  - As in original Chord

- **Used by e.g. Kademlia (distributed BitTorrent Tracker)**
9.1 Basic Chord Durability

• Advantages
  – Failure: usually, no additional copying of data needed
  – Rebuild intervals with neighbors only if critical
  – Requests can be answered by $k$ different nodes
    • Query load balancing possible

• Disadvantages
  – Less number of intervals as in original Chord
    • Solution: Virtual Servers
9.2 Load Balancing

• Load balancing goal:
  – Query and/or storage load should be distributed equally over all DHT nodes

• Common assumption
  – DHTs are naturally load-balanced
    • Storage load balancing due to good hash function
9.2 Load Balancing

• Assumption 1: **uniform key distribution**
  – Keys are generated uniformly by hash function

• Assumption 2: **equal data distribution**
  – Uniform keys will result in uniform data
  – Data is thus uniformly distributed

• Assumption 3: **equal query distribution**
  – Uniform keys will result in uniform queries
  – Each node has thus a similar query load

• But is this assumption justifiable?
9.2 Load Balancing

• Analysis of distribution of data using simulation

• Example
  – Parameters
    • 4,096 nodes
    • 500,000 documents
  – Optimum
    • ~122 documents per node
  – Some items are highly replicated due to popularity

• → No optimal distribution in Chord without load balancing
9.2 Load Balancing

- Number of nodes without storing any document
  - Parameters
    - 4,096 nodes
    - 100,000 to 1,000,000 documents

- Some nodes without any load

- Why is the load unbalanced?
- We need load balancing to keep the complexity of DHT management low
9.2 Load Balancing

• Definitions
  – DHT with $N$ nodes
  – **Optimally Balanced:**
    • Load of each node is around $\frac{1}{N}$ of the total load
  – A node is **overloaded** (or **heavy**)
    • Node has a significantly higher load compared to the optimal distribution of load
  – Else the node is **light**
9.2 Load Balancing

• **Load Balancing Algorithms**
  – **Problem**
    • Significant difference in the load of nodes
  – **There are several techniques to ensure an equal data distribution**
    – **Power of Two Choices**
      • (Byers et. al, 2003)
    – **Virtual Servers**
      • (Rao et. al, 2003)
    – **Thermal-Dissipation-based Approach**
      • (Rieche et. al, 2004)
    – **Simple Address-Space and Item Balancing**
      • (Karger et. al, 2004)
    – …
9.2 Load Balancing

• Algorithms

  – Power of Two Choices (Byers et. al, 2003)

  – Virtual Servers (Rao et. al, 2003)
• **Power of Two Choices**
  
  – **One hash function for nodes**
    
    • $h_0$
  
  – **Multiple hash functions for data**
    
    • $h_1, h_2, h_3, \ldots h_d$
  
  – **Two options**
    
    • Data is stored at one node only
    • Data is stored at one node & other nodes store a pointer
9.3 Power of Two Choices

- **Inserting Data** \(x\)
  - Results of all hash functions are calculated
    - \(h_1(x), h_2(x), h_3(x), \ldots, h_d(x)\)
  - Contact all \(d\) responsible nodes
    - Data is stored on the node with the lowest load
  - Alternative: other nodes store pointer
  - The owner of the item has to insert the document periodically
    - Prevent removal of data after a timeout (soft state)
9.3 Power of Two Choices

• Retrieving
  – Without pointers
    • Results of all hash functions are calculated
    • Request all of the possible nodes in parallel
    • One node will answer
  – With pointers
    • Request only one of the possible nodes.
    • Node can forward the request directly to the final node
9.3 Power of Two Choices

- **Advantages**
  - Simple

- **Disadvantages**
  - Message overhead for inserting data
  - With pointers
    - Additional administration of pointers lead to even more load
  - Without pointers
    - Message overhead for every search
9.4 Virtual Servers

• Algorithms
  – Power of Two Choices (Byers et. al, 2003)
  – Virtual Servers (Rao et. al, 2003)
9.4 Virtual Servers

• **Virtual Server**
  
  – Each *node* is responsible for several intervals
    
    • i.e. acts as multiple nodes
    • \( \log(n) \) virtual servers
    • "Virtual server"

[Diagram of a Chord Ring with servers connected]
9.4 Virtual Servers

- Each node is responsible for several intervals
  - **Load balancing** is achieved by **creating** or **transferring** virtual servers
    - Virtual servers take over responsibility for an arc and obtain copies of data
    - If a node is too heavy, it can transfer the virtual server to another node
- Different possibilities to change servers
  - One-to-one
  - One-to-many
  - Many-to-many
• **Rules** for transferring a virtual server
  – Transfer from *heavy node* to *light node*
  – The transfer of an virtual server should not make the receiving node heavy
    • Receiving node should have enough capacity
  – The transferred virtual server is the *lightest virtual server* that makes the heavy node light
    • Transfer as much as needed, but not more
  – If no single virtual server can make the node light, just transfer the heaviest one
    • In a second iteration, another virtual server can be transferred to another node
9.4 Virtual Servers

- Scheme: One-to-One
  - Light node picks a random ID
  - Contacts the node x responsible for it
  - Accepts load if x is heavy
9.4 Virtual Servers

- Scheme: One-to-Many
  - Light nodes report their load information to directories
  - Heavy node $H$ request information on light nodes from directory
    - $H$ contacts the light node which can accept the excess load directly

![Diagram showing the relationship between light nodes, directories, and heavy nodes.]

Distributed Data Management – Christoph Lofi – IfS – TU Braunschweig
9.4 Virtual Servers

- **Many-to-Many**
  - Heavy and light nodes rendezvous with directory
  - Directories periodically compute the transfer schedule and report it back to the nodes
    - Nodes just follow directory plan

Light nodes: $L_1, L_2, L_3, L_4, L_5$

Directories: $D_1, D_2$

Heavy nodes: $H_1, H_2$
• **Virtual Servers**
  
  – **Advantages**
    * Easy shifting of load
      – Whole Virtual Servers are shifted
  
  – **Disadvantages**
    * Increased administrative and messages overhead
      – Maintenance of all Finger-Tables
    * Much load is shifted
9.4 Virtual Servers

• Simulation
  – **Scenario**
    • 4,096 nodes
    • 100,000 to 1,000,000 documents
  – **Chord**
    • $M = 22$ bits
    • Consequently, $2^{22} = 4,194,304$ nodes and documents
  – **Hash function**
    • Sha-1 ($\bmod 2^m$)
    • random
  – **Analysis**
    • Up to 25 runs per test
9.4 Virtual Servers

**Without load balancing**

- Simple
- Bad load balancing

**Power of 2 Choices**

- Simple
- Lower load
- Nodes w/o load

**Virtual servers**

- No nodes w/o load
- Higher max. load than Power of Two Choices
• Stands for: **Lots Of Copies Keep Stuff Safe**
  – Goal: *disaster-proof long-term preservation* of digital content
  – Idea: distributing copies over the network will make access easy and keep material online, even in face of peer faults
  – [http://www.lockss.org](http://www.lockss.org)
    • HP Labs 1999

• Currently, many libraries world-wide participate in LOCKSS to preserve their digital content
  – Base motivation: digital content is part of the world heritage and should be protected and preserved
    • “...let us save what remains: not by vaults and locks which fence them from the public eye and use in consigning them to the waste of time, but by such a multiplication of copies, as shall place them beyond the reach of accident.” — Thomas Jefferson, February 18, 1791
• LOCKSS is not a traditional archive
  – Archives are for materials that are hard to replicate
    • i.e. original book from medieval ages
  – Archives sacrifice access to ensure preservation
    • e.g. disaster-proof underground archive

• LOCKSS ensures ubiquitous access and preservation of digitally replicable material
  – Allowing access puts preservation at risk, but risk can be minimized

• Central Question
  – How do you ensure that copies in the system are not compromised and never lost?
• **Design Goals of LOCKSS**

  – **Be affordable**
    • Cheap hardware
    • Open-source software
    • Low administration “appliance”

  – **Provide high data resilience and scalability**
    • Provide heavy replication resilient to attacks and disasters
    • Scale to enormous rates of publishing

  – **Allow access**
    • Allow search and access features
    • Conform to publishers access controls

  – Libraries take custody of content
Why is Long-Term Storage Hard?

- Large-scale disaster
- Human error
- Media faults
- Component faults
- Economic faults
- Organized attack
- Organizational faults
- Media/hardware obsolescence
- Software/format obsolescence
- Lost context/metadata
9.5 LOCKSS

- Solving the problem
  - Use a globally distributed P2P infrastructure
    - e.g. hosted by libraries
  - Allows for affordable cost models
    - Commodity hardware
    - Reduce on-going costs
  - **Replicate** content, break correlations between replicas
    - Geographic, administrative, platform, media, formats…
  - **Audit** replicas proactively to detect damage
    - Data must be accessible to do this cheaply!
  - Regularly **migrate** content to maintain usability
    - To new hardware, formats, keys…
  - Avoid external **dependencies**
    - E.g. vendor lock-in, DRM issues
  - Plan for data exit
• Exploit existing replication
  – **Testbed**: electronic *journals* in libraries
  – Many **libraries** subscribe to the same materials
  – Appliances used by libraries around the world
    • Cheap PC with some storage
    • Libraries maintain existing relationships with publishers
    • Materials are subscribed to be collected/preserved
    • Run a P2P audit/repair protocol between LOCKSS peers
    • Not a file sharing application
  – **Survive** or degrade gracefully in the face of attacks
    • Latent storage faults & sustained attacks
  – Make it hard to change consensus of population
• How does LOCKSS actually work?
  – The LOCKSS *audit/repair* protocol
  – A peer periodically *audits* its own content
    • To check its integrity
    • Calls an opinion poll on its content every 3 months
    • Gathers repairs from peers
  – Raises alarm when it suspects an attack
    • Correlated failures
    • IP address spoofing
    • System slowdown
• Sampled Opinion Poll
  – Each peer holds a poll for each document
    • Reference list of peers it has discovered
    • History of interactions with others (balance of contributions)
  – Periodically (faster than rate of storage failures)
    • Poller takes a random sample of the peers in its reference list
    • Invites them to vote: send a hash of their replica
  – Compares votes with its local copy
    • Overwhelming agreement (>70%) ➔ Sleep blissfully
    • Overwhelming disagreement (<30%) ➔ Repair
    • Too close to call ➔ Raise an alarm
  – Repair: peer gets pieces of replica from disagreeing peers
    • Re-evaluates the same votes
  – Every peer is both poller and voter
• Most replicas the same
  – No alarms

• Some replicas corrupted
  – Alarms very likely
  – To achieve full corruption:
    • Adversary must pass through “moat” of alarming states
    • Damaged peers vote with undamaged peers
    • Rate limitation helps

Adversary’s Intention

Probability of avoiding alarms

All Good

All Bad
• Probability of Irrecoverable Damage

Preservation succeeds for up to 35% subversion

- For powerful attacker (unlimited CPU/identities)
- Attacking for 30 years
9.5 Ocean Store

- Application: build a **P2P cloud storage**
  - Improve availability through wide replication
  - Untrusted decentralized infrastructure

- **OceanStore**: provide long-time available data
  - Layered architecture
    - Inner ring holds committed data
      - uses byzantine agreement to secure data
    - Outer nodes contain normal users
  - Target is global scale data access
9.5 Ocean Store

• Ubiquitous Devices $\Rightarrow$ Ubiquitous Storage
  – Consumers of data move, change from one device to another, work in cafes, cars, airplanes, the office, etc.

• Properties required for OceanStore storage
  – **Strong Security**
    • data encrypted in the infrastructure
    • resistance to monitoring and denial of service attacks
  – **Coherence**
    • too much data for naïve users to keep coherent “by hand”
– **Automatic replication management and optimization**
  - Huge quantities of data cannot be managed manually

– **Simple and automatic recovery from disasters**
  - Probability of failure increases with size of system

– **Utility model**
  - World-scale system requires cooperation across administrative boundaries
9.5 Ocean Store

• Everyone’s Data, One Big Utility
  – Shared Cloud Storage
  – “The data is just out there”

• Separate information from location
  – Locality is an only an optimization
  – Wide-scale coding and replication for durability

• All information is globally identified
  – Unique identifiers are hashes over names & keys
  – Single uniform lookup interface
  – No centralized namespace required
• OceanStore assumptions
  – **Untrusted Infrastructure**
    • OceanStore is mainly made up of untrusted components
      – Use only cyphertext within the infrastructure
    • Information must not be “leaked” over time
  – **Mostly well-connected**
    • Data producers and consumers are connected to a high-bandwidth network most of the time
    • Exploit multicast for quicker consistency when possible
  – **Promiscuous Caching**
    • Data may be cached anywhere, anytime
  – **Trusted party** is responsible for keeping up service
    • Probably won’t disconnect and is probably not malicious
Storage Issues

- Where is persistent information stored?
  - Wanted: geographic independence for availability, durability, and freedom to adapt to circumstances

- How is it protected?
  - Wanted: encryption for privacy, signatures for authenticity, and Byzantine commitment for integrity

- Can we make it indestructible?
  - Wanted: redundancy with continuous repair and redistribution for long-term durability

- Is it hard to manage?
  - Wanted: automatic optimization, diagnosis and repair
9.5 Ocean Store

• Naming and Data Location
  – Requirements:
    • System-level names should help to authenticate data
    • Route to nearby data without global communication
    • Don’t inhibit rapid relocation of data
  – Approach: Two-level search with embedded routing
    • Underlying namespace is flat and built from secure cryptographic hashes (160-bit SHA-1)
    • Search process combines quick, probabilistic search with slower guaranteed search
    • Long-distance data location and routing are integrated
      – Every source/destination pair has multiple routing paths
      – Continuous, on-line optimization adapts for hot spots, denial of service, and inefficiencies in routing
OceanStore Approach:

- Operations-based interface using conflict resolution
  - Modeled after Xerox Bayou ⇒ updates packets include:
    - Predicate/update pairs which operate on encrypted data
  - Use of oblivious function techniques to perform this update
  - Use of incremental cryptographic techniques

- User signs Updates and trusted party signs commits

- Committed data multicast to clients
Epidemic Dissemination
Multicast Dissemination

Trusted Party
9.5 Ocean Store

• **Oceanstore: State of the Art**
  – **Techniques for protecting metadata**
    • Uses encryption and signatures to provide protection against substitution attacks
  – **Working scheme that can do some forms of conflict resolution directly on encrypted data**
    • Uses new technique for searching on encrypted data.
    • Can be generalized to perform optimistic concurrency, but at cost in performance and possibly privacy
  – **Byzantine assumptions for update commitment**
    • Signatures on update requests from clients
      – Compromised servers are unable to produce valid updates
      – Uncompromised second-tier servers can make consistent ordering decision with respect to tentative commits
9.5 Ocean Store

• High-Availability and Disaster Recovery
  – Requirements
    • Handle diverse, unstable participants in OceanStore
    • Mitigate denial of service attacks
    • Eliminate backup as independent (and fallible) technology
    • Flexible “disaster recovery” for everyone
  – OceanStore Approach
    • Use of erasure-codes to provide stable storage for archival copies and snapshots of live data
    • Version-based update for painless recovery
    • Continuous introspection repairs data structures and degree of redundancy
9.5 Ocean Store

- Archival Dissemination of Fragments
• **Automatic Maintenance**
  
  – **Byzantine Commitment for inner ring:**
    - Can tolerate up to 1/3 faulty servers in inner ring
      - Bad servers can be arbitrarily bad
      - Cost $\sim n^2$ communication
    - Continuous refresh of set of inner-ring servers
      - Proactive threshold signatures
      - Use of Tapestry $\Rightarrow$ membership of inner ring unknown to clients
  
  – **Secondary tier self-organized into overlay dissemination tree**
    - Use of Tapestry routing to suggest placement of replicas in the infrastructure
    - Automatic choice between update vs. invalidate
10.0 Special Purpose Database

10.1 Trade-Offs
   - CAP Theorem
   - BASE transactions

10.2 Showcase: Amazon Dynamo