Encrypted Distributed Hash Tables

Archita Agarwal, Seny Kamara
Chord: A Scalable Peer-to-peer Lookup Protocol for ... - MIT PDOS
by I Stoica Cited by 13655 Related articles
presents Chord, a distributed lookup protocol that addresses this problem. Chord provides .... tion of blocks. The distributed hash table uses Chord to identify.

Bigtable: A Distributed Storage System for Structured Data
https://dl.acm.org/citation.cfm?id=1365816
by F Chang - 2008 - Cited by 6119 Related articles
Sep 1, 2017 - Citation Count: 432 Downloads (cumulative): 21,596 ... ACM Transactions on Computer Systems (TOCS) TOCS Homepage archive. Volume 26 Issue 2, June 2008. Article No. 4. ACM New York, NY, USA table of contents ...

Dynamo: Amazon's Highly Available Key-value Store
by G DeCandelo - 2007 Cited by 4704 Related articles
Oct 14, 2007 - This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios.
Distributed Hash Tables
DHT
DHT

- Decentralised Systems
- Distribute \((l, v)\) pairs to nodes
DHT

- Decentralised Systems
- Distribute \((\ell, v)\) pairs to nodes
- Supports \(\text{Get}(\ell), \text{Put}(\ell, v)\) operations
Decentralised Systems

- Distribute \((l, v)\) pairs to nodes
- Supports Get\((l)\), Put\((l, v)\) operations
- Overlay network
- Routing protocol
Classic Applications of DHTs

Democratizing content publication with Cori

Michael J. Freedman, Eric Freudenthal, David Mazieres
New York University

Faster Content Access in KAD

Moritz Steiner, Damiano Carra, and Ernst W. Biersack
Eurecom, Sophia-Antipolis, France

Squirrel: A decentralized peer-to-peer web cache

Silaram Iyer
Rice University
6100 Main Street, MS-132
Houston, TX 77005, USA
ssiyer@cs.rice.edu

Antony Rowstron
Microsoft Research
7 J J Thomson Close
Cambridge, CB3 0FB, UK
anstr@microsoft.com

Peter Druschel
Rice University
6100 Main Street
Houston, TX 77005, USA
druschel@rice.edu

ABSTRACT
This paper presents a decentralized, peer-to-peer web cache called Squirrel. The key idea is to enable web browsers on desktop machines to share their local caches, to form an efficient and scalable web cache, without the need for dedicated hardware and the associated administrative cost. We propose and evaluate decentralized web caching algorithms for Squirrel, and discover that it exhibits performance comparable to a centralized web cache in terms of hit ratio, bandwidth usage and latency. It also achieves the benefits of decentralization, such as being scalable, self-organizing and resilient to node failures, while avoiding low overhead on the participant.

There is substantial literature in web caching [3, 6, 9, 20, 23, 24] and content distribution [4]. This paper demonstrates a scalable, efficient and self-organizing system. We present an algorithm (Squirrel) that is able to achieve these goals without the need for dedicated hardware and without imposing a high overhead on the participants.

A DHT-based Infrastructure for Content-based Publish/Subscribe Services* 

Xiaoyu Yang and Yiming Hu
Department of Electrical and Computer Engineering
University of Cincinnati, Cincinnati, OH 45221, USA
{yangxu,yhu}@ececs.uc.edu

Abstract. A DHT-based infrastructure is presented for content-based publish/subscribe services.

SCAN: A Dynamic, Scalable, and Efficient Content Distribution Network

Yan Chen, Randy H. Katz and John D. Kubiatowicz
Computer Science Division, University of California at Berkeley

Abstract. We present SCAN, the Scalable Content Access Network. SCAN is a DHT-based system designed to provide efficient content distribution for large-scale networks. We discuss the design, implementation, and performance of SCAN, and compare it to existing systems.

* The authors acknowledge the support of the National Science Foundation under grant CCR-0110746.
Classic Applications of DHTs

Content Delivery Networks

P2P File Sharing
Classic Applications of DHTs

- Content Delivery Networks
- P2P File Sharing
- Distributed File Systems

Wide-area cooperative storage with CFS

Frank Dabek, M. Frans Kaashoek, and Ion Stoica
MIT Laboratory for Computer Science
http://pdcgroup.csail.mit.edu

Ivy: A Read/Write Peer-to-Peer File System

Athicha Muthitacharoen, Robert Morris, Thomer M.
{athicha, rtm, thomer, benjie}@ics.mit.edu
MIT Laboratory for Computer Science
Cambridge, MA 02139.

Pond: the OceanStore Prototype

Sean Rhea, Patrick Eaton, Dennis Geels,
Hakim Weatherspoon, Ben Zhao, and Ion Stoica
University of California, Berkeley
{srahea, eaton, geels, hweatherspoon, benzho}@cs.berkeley.edu

PAST: A large-scale, persistent peer-to-peer storage utility

Peter Druschel
Rice University
Houston, TX 77005-1892, USA
druschel@cs.rice.edu

Antony Rowstron
Microsoft Research Ltd.
Cambridge, CB2 3NH, UK
antr@microsoft.com

Abstract

P2P File Sharing

While PAST offers persistent storage services, its access semantics differ from that of a conventional filesystem. Files stored in PAST are associated with a field that

Abstract

This paper describes PAST, a large-scale, Internet-based, storage system designed for distributed environments. PAST provides a high-level interface for accessing

Abstract

OceanStore is an Internet-scale, persistent data store designed for incremental scalability, secure sharing, and long-term durability. Pond is the OceanStore prototype: it contains many of the features of a complete system, including

Abstract

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

PAST: A large-scale, persistent peer-to-peer storage utility

Peter Druschel
Rice University
Houston, TX 77005-1892, USA
druschel@cs.rice.edu

Antony Rowstron
Microsoft Research Ltd.
Cambridge, CB2 3NH, UK
antr@microsoft.com

While PAST offers persistent storage services, its access semantics differ from that of a conventional filesystem. Files stored in PAST are associated with a field that

While PAST offers persistent storage services, its access semantics differ from that of a conventional filesystem. Files stored in PAST are associated with a field that

While PAST offers persistent storage services, its access semantics differ from that of a conventional filesystem. Files stored in PAST are associated with a field that
Classic Applications of DHTs

1. Content Delivery Networks
2. Distributed File Systems
3. Key-Value Stores
4. P2P File Sharing

---

**Dynamo** Amazon’s Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Sivasubramanian Sivakumar, Peter Vosshall and Werner Vogels

*Amazon.com*

**ABSTRACT**

Reliability at massive scale is one of the biggest challenges at Amazon.com, one of the largest e-commerce websites in the world. Even the slightest outage has consequences and impacts customer trust. Dynamo, which is implemented on top of an infrastructure of commodity servers and network components located around the world. At this scale, small and large failures are continuous, and the way persistent state is maintained drives the reliability and availability of software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that supports services to provide an “always-on” experience to users. To achieve this level of availability, Dynamo sacrificed on a primary partition failure scenario, allowing it to reduce concurrency and increase availability. Dynamo’s high availability, reliability, and access control is achieved by providing a novel interface for developers to use.

**Categories and Subject Descriptors**

D.4.2 [Databases]: {Distributed Databases}

**Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

**ACM Reference Format**

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Sivasubramanian Sivakumar, Peter Vosshall and Werner Vogels. 2010. Dynamo: Amazon’s Highly Available Key-value Store.

---

**Serving Large-scale Batch Computed Data with Project Voldemort**

Rohan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, Sam Shah

**Abstract**

Current serving systems lack the ability to bulk load massive in-mutable data sets without affecting serving performance. The performance degradation is largely due to CPU and memory contention and modification as CPU and memory contention and modification. We describe Voldemort, a system that can be used to serve this data. Voldemort is a key-value storage system that acts as a highly scalable and fault-tolerant backend for large-scale batch computing. Voldemort is designed to support large-scale batch computing by providing high availability, scalability, and fault tolerance. Voldemort is implemented using a distributed system of nodes, each of which consists of a collection of processes running on a single machine. Each node in the Voldemort system is responsible for a subset of the key-value space, and each node maintains a local copy of the data. Voldemort provides a client API that allows clients to easily access the data stored in Voldemort.

---

You May Know feature on LinkedIn runs on hundreds of terabytes of offline data daily to make these predictions. Due to the dynamic nature of the social graph, this derived data changes extremely frequently—requiring an almost complete refresh and bulk load of the data, while continuing to serve existing traffic with minimal additional latency. Naturally, this batch update should be highly efficient to minimize the time and resources required to perform updates.

---

LinkedIn
Classic Applications of DHTs

- Content Delivery Networks
- Distributed File Systems
- P2P File Sharing
- Key-Value Stores
- NoSQL Databases

Cassandra: A Decentralized Structured Storage System

Avinash Lakshman, Prashant Malik
Facebook

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3h,tushar,flies,gruber}@google.com
Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Fi-
Recent Application of DHTs

- Off-Chain Storage in Blockchains
- Content Delivery Networks
- P2P File Sharing
- Distributed File Systems
- Key-Value Stores
- NoSQL Databases
Recent Application of DHTs

- Off-Chain Storage in Blockchains
- Content Delivery Networks
- P2P File Sharing
- Distributed File Systems
- Key-Value Stores
- NoSQL Databases
- Encryption

- Ethereum
- Enigma
- STORJ.IO
- Filecoin

Abstract

Recent advancements in distributed hash table (DHT) technology have enabled a wide range of applications, including:

- Content Delivery Networks (CDNs) for efficient content distribution
- Peer-to-Peer (P2P) file sharing systems for decentralized file exchange
- Distributed file systems for scalable storage solutions
- Key-value stores for fast data lookup and retrieval
- NoSQL databases to handle unstructured data

These DHT technologies have also been applied in innovative ways, such as:

- Off-chain storage in blockchains to improve scalability and privacy
- Encryption protocols for secure data transmission and storage

These developments are further explored in the following sections, highlighting the latest research and applications in the field.
Simple Standard Scheme

**Put(ℓ, v)**
- Apply PRF on label
- Encrypt value
- Store in DHT

**Get(ℓ)**
- Apply PRF on label
- Fetch value from DHT using pseudorandom label
Q: What is the security of this standard scheme?
Q1: What information is learnt by Adversary about these pairs?

Q2: Does it only learn information about the pairs it stores?

Maybe
Relation to Structured Encryption

Learns about all the pairs
Q1: What information is learnt by the adversary about the client’s data/queries?

Q2: Does it only learn information about the pairs it stores?

NO

Infer a good approximation of the total number of pairs!

Total pairs adv. holds: $m$

Total expected pairs: $\sim mn/t$

if DHTs are load balanced

Analyzing leakage in Distributed Systems is tricky!
Formalize the use of end-to-end encryption in DHTs
- Chord DHT
- Formalize DHTs
- Formalize EDHTs
- Syntax
- Security
- Analyze Standard Scheme
- Extend to Transient Setting
- Takeaways & Open Questions
Chord DHT

- Address Space: $A$
- Two hash functions:
  - $H_1$: hashes node ids to addresses
  - $H_2$: hashes labels to addresses
- $\text{server}(l) = \text{successor}(H_2(l))$
Chord DHT

- Address Space: $A$
- Two hash functions:
  - $H_1$: hashes node ids to addresses
  - $H_2$: hashes labels to addresses
- $\text{server}(\ell): \text{successor}(H_2(\ell))$
- $\text{route}(a_1, a_2)$:
  - logarithmic sized routing tables
  - logarithmic sized paths
Chord DHT

- **Address Space**: A
- **Two hash functions**:
  - $H_1$: hashes node ids to addresses
  - $H_2$: hashes labels to addresses
- **server($l$): successor($H_2(l)$)**
- **route($a_1$, $a_2$):**
  - logarithmic sized routing tables
  - logarithmic sized paths
Chord DHT

- Address Space: $A$
- Hash functions:
  - $H_1$: hashes node ids to addresses
  - $H_2$: hashes labels to addresses
- $server(l)$: $successor(H_2(l))$
- $route(a_1, a_2)$:
  - logarithmic sized routing tables
  - logarithmic sized paths

Overlay param

Allocation param

Overlay param

Determined by $H_1, H_2$
Chord: Visible addresses

- \( \text{Vis}(N) \): set of all addresses s.t. if \( H_2(\ell) = a \) then either
  - \( \text{server}(\ell) = N \)
  - \( N \in \text{route}(a) \)
Chord: Visible addresses

- $\text{Vis}(N)$: set of all addresses s.t. if $H_2(\ell) = a$ then either
  - $\text{server}(\ell) = N$
  - $N \in \text{route}(a)$
Chord: Visible addresses

- $\text{Vis}(N)$: set of all addresses s.t. if $H_2(\ell) = a$ then either
  - $\text{server}(\ell) = N$
  - $N \in \text{route}(a)$
Chord DHT
Formalize DHTs
Formalize EDHTs
Syntax
Security
Analyze Standard Scheme
Extend to Transient Setting
Takeaways & Open Questions
Formalizing DHTs

\[ \text{DHT} = (\text{Overlay, Alloc, Daemon, Put, Get}) \]
DHT = (Overlay, Alloc, Daemon, Put, Get)

- Executed only once
- At the time of setup
- Overlay outputs $\omega$
- Alloc outputs $\psi$
Formalizing DHTs

DHT = (Overlay, Alloc, Daemon, Put, Get)

- Executed only once
- At the time of setup
- Overlay outputs $\omega$
- Alloc outputs $\psi$
- Executed by all nodes
- All the time
- Sends/receives messages
- Stores/retrieves (label, value) pairs
Formalizing DHTs

DHT = (Overlay, Alloc, Daemon, Put, Get)

- Executed only once
- At the time of setup
- Overlay outputs \( \omega \)
- Alloc outputs \( \psi \)
- Executed by all nodes
- All the time
- Sends/receives messages
- Stores/retrieves (label, value) pairs
- Executed by client
- To store/retrieve (label/value) pair in/from network
Formalizing DHTs

DHT = (Overlay, Alloc, Daemon, Put, Get)

- Executed only once
- At the time of setup
- Overlay outputs $\omega$
- Alloc outputs $\psi$
- Executed by all nodes all the time
- Sends/receives messages
- Stores/retrieves (label, value) pairs
- Executed by client
- To store/retrieve (label/value) pair in/from network

addr $\omega : N \rightarrow A$

server $\omega,\psi : L \rightarrow A$

route $\omega : A \times A \rightarrow 2^A$
Properties of DHTs
Properties of DHTs

P1: Balance

P2: Non-committing allocations

“And if elected, I promise to keep making promises.”
P1: Balance

- Overlay $\omega$ is $\varepsilon$-balanced if $\forall$ labels $l$, and all nodes $N$
  - $\Pr[\text{server}(l) \in \text{Vis}(N)] \leq \varepsilon$
  - Prob over choice of $\psi$

- A DHT is $(\varepsilon, \delta)$-balanced if
  - $\Pr[\omega \text{ is } \varepsilon\text{-balanced}] \geq 1 - \delta$
  - Prob over choice of $\omega$
\[ \Pr[\text{server}(\ell) \in \text{Vis}(N)] = \Pr[\text{server}(\ell) = N] + \Pr[N \in \text{route}(\text{server}(\ell))] \]

\(\propto\) length of arc of \(N\)

with high prob max arc \(\leq (4 |A| \log n)/n\)

- \(\Pr[\text{server}(\ell) = N']\)
- \(\Pr[N \text{ on log sized path to } N']\)
Balance of Chord

**Theorem:**

Chord is \((\varepsilon, \delta)\)-balanced for

\[
\varepsilon = \frac{(4 \log n) + 4n \log^2 n}{n} \quad \text{and} \quad \delta = \frac{1}{n}
\]

- If \(|A| = 2^{512} \Rightarrow n^2 \log n < |A|\), even for \(n = 2^{250} \Rightarrow \varepsilon = O\left( \frac{\log n}{n} \right)\)
be able to change/program $\psi$

- given a label $\ell$ and an address $a$
- set $\psi(\ell) = a$
Outline

- Chord DHT
- Formalize DHTs
- Formalize EDHTs
- Syntax
- Security
- Analyze Standard Scheme
- Extend to Transient Setting
- Takeaways & Open Questions
Formalizing EDHTs : Syntax

EDHT = (Gen, Overlay, Alloc, Daemon, Put, Get)
Formalizing EDHTs : Syntax

EDHT = (Gen, Overlay, Alloc, Daemon, Put, Get)

Same as before
Formalizing EDHTs: Syntax

EDHT = (Gen, Overlay, Alloc, Daemon, Put, Get)

- Executed by Client
- Generates reqd. keys for client

Same as before
EDHTs Security

Real

Ideal

\[ T \] Executes Overlay(), Alloc()

\[ Z \]

\[ A \]  

\[ C \]  

I  

op
EDHTs Security

\[ T \text{ Executes } \text{Overlay(), Alloc()} \]

\[ A \]

\[ \mathcal{Z} \]

\[ C \]

\[ F \]

\[ \text{Put}(\ell, v): \text{Sets } DX[\ell] := v \]

\[ \text{Get}(\ell): \text{Outputs } DX[\ell] \]

Real $\approx$ Ideal
- Chord DHT
- Formalize DHTs
- Formalize EDHTs
- Syntax
- Security
- Analyze Standard Scheme
- Extend to Transient Setting
- Takeaways & Open Questions
Standard Scheme: Construction

**Gen($1^k$)**
- Sample $K_1 \leftarrow \{0, 1\}^k$
- $K_2 \leftarrow \text{SKE.Gen}(1^k)$
- Output $(K_1, K_2)$

**Put$(K, \ell, v)$**
- $K = (K_1, K_2)$
- $t = F_{K_1}(\ell)$
- $e = \text{SKE.Enc}_{K_2}(v)$
- DHT.Put($t$, $e$)

**Get$(K, \ell)$**
- $K = (K_1, K_2)$
- $t = F_{K_1}(\ell)$
- $e \leftarrow \text{DHT.Get}(t)$
- $v \leftarrow \text{SKE.Dec}_{K_2}(e)$
Understanding Leakage

\[ \omega_1 \quad \omega_2 \quad \omega_3 \quad \omega_4 \]
Q: Is there any gain over STE leakage?

YES

Very unlikely*
Understanding Leakage

DHT is \((\varepsilon, \delta)\)-balanced if

\[
\Pr[\omega \text{ is } \varepsilon\text{-balanced}] \geq 1 - \delta
\]

\(
\Pr[\text{label being visible to a node}] \leq \varepsilon
\)

leaks \(q_{eq}(t)\) with probability \(\min(1, t \cdot \varepsilon)\)

\(L\) is probabilistic

affected by balancing properties of DHT

Very unlikely*

Sampling a “bad” overlay is unlikely
Standard Scheme: Security

**Th:** If DHT is $(\epsilon, \delta)$-balanced and has non-committing overlays, then EDHT is $L_\epsilon$-secure with prob at least $1 - \delta - \text{negl}(k)$.
Challenges in Proof

\[ \mathcal{L} \]

needs to generate leakages compatible with \( \omega \)

Two options:

- leak all the queries
- \( \mathcal{L} \) generates \( \omega \)
- Chord DHT
- Formalize DHTs
- Formalize EDHTs
- Syntax
- Security
- Analyze Standard Scheme
- Extend to Transient Setting
- Takeaways & Open Questions
Transient Setting

Nodes can leave/enter the network
DHT = (Overlay, Alloc, Daemon, Put, Get, Leave, Join)

EDHT = (Gen, Overlay, Alloc, Daemon, Put, Get, Leave, Join)

❖ Run by node wishing to leave the network
❖ Run by node wishing to join the network
Leave/Join in Chord
Security: Transient EDHTs

$\text{Put}(l, v): \text{Sets } DX[l] := v$

$\text{Get}(l): \text{Outputs } DX[l]$

Real

Ideal

$\mathcal{T}$ executes Overlay(), Alloc()

$Z$ can now also issue leave/join requests

$L(DX, \text{op})$

Real $\approx$ Ideal
Properties of DHTs

P1: Balance

Stronger notion

- A DHT is \((\varepsilon, \delta)\)-balanced if for all active nodes \(C\)
  - \(\Pr[\land (\omega, C) \text{ is } \varepsilon\text{-balanced}] \geq 1 - \delta\)

w/ prob 1-\(\delta\) the sampled overlay is balanced for all nodes \(C\)

P2: Non-committing allocations

Same as before

"And if elected, I promise to keep making promises."
Additional pairs become visible during leave/join

which pairs to leak??

leaks \( qeq(l) \) of all the previous pairs
Transient Standard Scheme: Security

Th: If transient DHT is \((\varepsilon, \delta)\)-balanced and has non-committing overlays, then transient EDHT is \(L_\varepsilon\)-secure with prob at least \(1 - \delta - \text{negl}(k)\)
Outline

- Chord DHT
- Formalize DHTs
- Formalize EDHTs
- Syntax
- Security
- Analyze Standard Scheme
- Extend to Transient Setting
- Takeaways & Open Questions
Expected Leakage Analysis
- Earlier: leakage functions were deterministic
- Now: probabilistic

Co-design distributed systems with reqd. crypto

Building secure distributed systems can be tricky
- Intuitions are not always right

Distributing data can help in leakage suppression
Tighter analysis of Transient Chord

Study of $(\epsilon, \delta)$ of other DHTs
  - Kademlia, Koord

Design other EDHTs

Security in UC setting
Thank you