OpenDHT: A Shared, Public DHT Service

Sean C. Rhea

OASIS Retreat

January 10, 2005

Joint work with Brighten Godfrey, Brad Karp, Sylvia Ratnasamy, Scott Shenker, Ion Stoica and Harlan Yu

Distributed Hash Tables (DHTs)

- Introduced four years ago
  - Original context: peer-to-peer file sharing
- Idea: put/get as a distributed systems primitive
  - Put stores a value in the DHT, get retrieves it
  - Just like a local hash table, but globally accessible
- Since then:
  - Good implementations available (Bamboo, Chord)
  - Dozens of proposed applications

DHT Applications

- Storage systems
  - file systems: OceanStore (UCB), Past (Rice, MSR), CFS(MIT)
  - enterprise backup: hivecache.com
  - content distribution networks: BSlash(Stanford), Coral (NYU)
  - cooperative archival: Venti-DHASH (MIT), Pastiche (UMich)
  - web caching: Squirrel (MSR)
  - Usenet DHT (MIT)

DHT Applications

- Storage systems
- Indexing/naming services
  - Chord-DNS (MIT)
  - OpenDHT (Intel, UCB)
  - pSearch (HP)
  - Semantic Free Referencing (ICSI, MIT)
  - Layered Flat Names (ICSI, Intel, MIT, UCB)

DHT Applications

- Storage systems
- Indexing/naming services
- DB query processing
  - PIER (UCB, Intel)
  - Catalogs (Wisconsin)

DHT Applications

- Storage systems
- Indexing/naming services
- DB query processing
- Internet data structures
  - SkipGraphs (Yale)
  - PHT (Intel, UCSD, UCB)
  - Cone (UCSD)

DHT Applications

- Storage systems
- Indexing/naming services
- DB query processing
- Internet data structures
- Communication services
  - i3 (UCB, ICSI)
  - multicast/streaming: SplitStream, MCAN, Bayeux, Scribe, …

Deployed DHT Applications

- Overnet
  - Peer-to-peer file sharing
  - 10,000s of users
- Coral
  - Cooperative web caching
  - Several million requests per day

Why the discrepancy between hype and reality?

A Simple DHT Application: FreeDB Cache

- FreeDB is a free version of the CD database
  - Each disc has a (mostly) unique fingerprint
  - Map fingerprints to metadata about discs
- Very little data: only 2 GB or so
  - Trick is making it highly available on the cheap
  - Existing server: 4M reqs/week, 48 hour outage
- A perfect DHT application
  - One node can read DB, put each entry into DHT
  - Other nodes check DHT first, fall back on DB

Deploying the FreeDB Cache

- Download and familiarize self with DHT code
- Get a login on a bunch ($\approx$100) of machines
  - Maybe convince a bunch of friends to do it
  - Or, embed code in CD player application
  - PlanetLab, if you're really lucky
- Create monitoring code to keep up and running
  - Provide a service for clients to find DHT nodes
- Build proxy to query DHT before DB
- After all this, is performance even any better?
  - Is it any wonder that no one is deploying DHT apps?

An Alternative Deployment Picture

- What if a DHT was already deployed?
  - How hard is it to latch onto someone else's DHT?
- Still have build proxy to query DHT before DB
- After that, go direct to measuring performance
  - Don't have to get login on a bunch of machines
  - Don't have to build infrastructure to keep it running
- Much less effort to give it a try

OpenDHT

- Insight: *a shared DHT would be really valuable*
  - Could build/deploy FreeDB cache in a day
  - Dumping DB into DHT: $\approx$ 100 semicolons of C++
  - Proxy: 58 lines of Perl
- But it presents a bunch of research challenges
  - Traditional DHT APIs aren't designed to be shared
    - Every application's code must be present on every DHT node
    - Many traditional DHT apps modify the DHT code itself
  - A shared DHT must isolate applications from each other
    - Clients should be able to authenticate values stored in DHT
    - Resource allocation between clients/applications

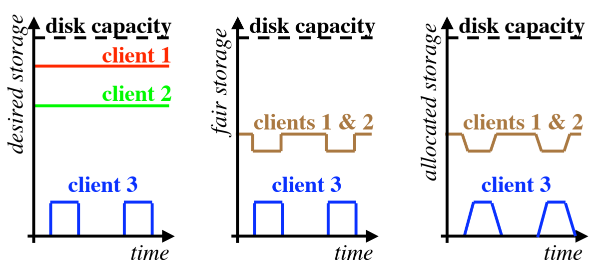
Protecting Against Overuse

- PlanetLab has a 5 GB per-slice disk quota
  - "But any *real* deployment will be over-provisioned."
- Peak load may be much higher than average load
  - A common problem for web servers, for example
- Malicious users may deny service through overuse
  - In general, can't distinguish from enthusiastic users
- Research goals:
  - Fairness: stop the elephants from trampling the mice
  - Utilization: don't force the elephants to become mice

Put/Get Interface Assumptions

- Make client code and garbage collection easy
  - Puts have a time-to-live (TTL) field
  - DHT either accepts or rejects puts "immediately"
  - If accepted, must respect TTL; else, client retries
- Accept based on fairness and utilization
  - Fairness could be weighted for economic reasons
- All decisions local to node
  - No global fairness/utilization yet
  - Rewards apps that balance puts, helps load balance

Fair Allocation Example

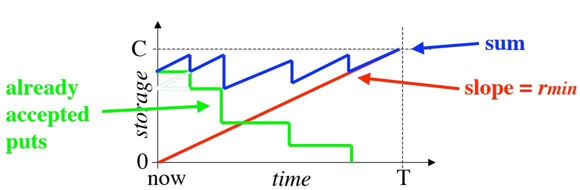


Starvation

- A motivating example
  - Assume we accept 5 GB of puts in very little time
  - Assume all puts have maximum TTL
- Result: starvation
  - Must hold all puts for max TTL, and disk full
  - Can't accept new puts until existing ones expire
- Clearly, this hurts fairness
  - Can't give space to new clients, for one thing

Preventing Starvation

- Fairness: must be able to adapt to changing needs
  - Guarantee storage frees up as some minimum rate, $r_{min} = C/T$
  - T is maximum TTL, C is disk capacity
- Utilization: don't rate limit when storage plentiful



Efficiently Preventing Starvation

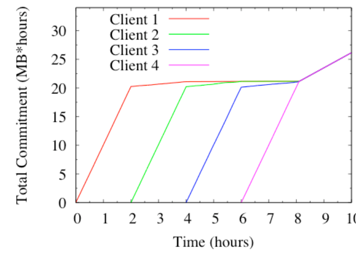
- Goal: before accept put, guarantee sum $\leq C$
- Naïve implementation
  - Track values of sum in array indexed by time
  - $O(T/\Delta t)$ cost: must update sum for all time under put
- Better implementation
  - Track inflection points of sum with a tree
  - Each leaf is an inflection point (time, value of sum)
  - Interior nodes track max value of all children
  - $O(\log n)$ cost: where n is the number of puts accepted

Fairly Allocating Put Rate

- Another motivating example
  - For each put, compute sum, if ≤ C, accept
  - Not fair: putting more often gives more storage
- Need to define fairness
  - Critical question: fairness of *what* resource?
  - Choice: storage over time, measured in bytes × seconds
  - 1 byte put for 100 secs same as 100 byte put for 1 sec
  - Also call these “commitments”

Candidate Algorithm

- Queue all puts for some small time, called a slot
  - If max put size is m, a slot is m/r_{min} seconds long
- At end of slot, in order of least total commitments:
  - If sum ≤ C for a put, accept it
  - Otherwise, reject it
- Result:
  - Starvation

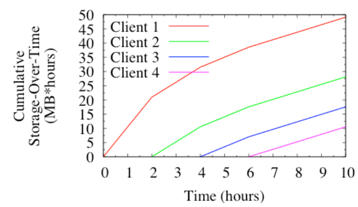


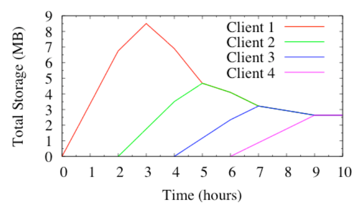
Preventing Starvation (Part II)

- Problem: we only prevented global starvation
  - Individual clients can still be starved periodically
- Solution: introduce *use-it-or-lose-it* principle
  - Don’t allow any client to fall too far behind
- Easy to implement
  - Introduce a minimum total commitment, s_{sys}
  - After every accept, increment client commitment, s_{client}, and s_{sys} both
  - When ordering puts, compute

    effective $s_{client} = \max(s_{client}, s_{sys})$

Revised Algorithm Performance





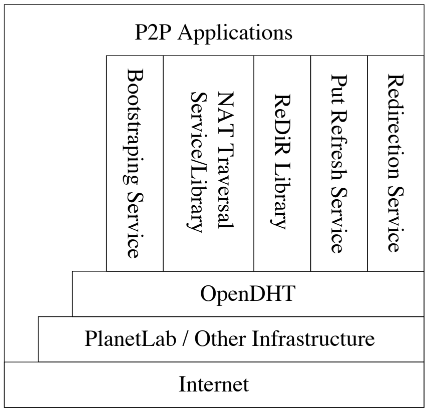
Fair Storage Allocation Notes

- Also works with TTL/size diversity
  - Not covered here
- Open Problem 1: can we remove the queuing?
  - Introduces a delay of 1/2 slot on average
  - Working on this now, but no firm results yet
- Open Problem 2: how to write clients?
  - How long should a client wait to retry rejected put?
  - Can it redirect the put to a new address instead?
  - Do we need an explicit refresh operation?

Longer Term Future Work

- OpenDHT makes a great common substrate for:
  - Soft-state storage
  - Naming and rendezvous
- Many P2P applications also need to:
  - Solve the bootstrap problem
  - Traverse NATs
  - Redirect packets within the infrastructure (as in *i3*)
  - Refresh puts while intermittently connected
- We need *systems software* for P2P

A System Architecture for P2P



For more information, see
http://opendht.org/