Hashing And Fingerprinting

11 February 2005

William Josephson
wkj@cs.princeton.edu
Motivation: Technology Trends

• Magnetic Disks (from last time):
  – Exponential increase in disk capacity
  – By comparison, small increases in angular velocity (\(\sim 4x\) since 1980)
  – Modest improvements in ballistic seek and settle times

• Wide-Area Networks:
  – Last five years have seen demise of the analog modem
  – Backbone bandwidth has increased rapidly
  – Prospect for comparable increase in bit rate over last mile slimmer
  – Latency on ISDN/DSL/SONET unlikely to improve much
  – Long latency and high drop rates common on the WAN
    * It is often faster to get to New York via Boston on Internet2
    * 1000ms RTTs are common between Princeton and the Internet
Motivation: Applications

- Bandwidth Reduction (LBFS)
  - How can I use my data over a long, thin pipe?
- Duplicate Elimination (Venti)
  - What happens if I save every version of every document?
- Naming in Distributed Systems (Chord)
  - What can I do to avoid the scourge of centralized name services?
- Similarity Search (Udi Manber)
  - Yikes! My disk is too big: how can I find anything?
- Naming of Automatically Generated Structured Data (Andrei Broder)
  - How can I collect statistics, type-check modules efficiently, & so on...
Hashing in Theory

• Generally we use a hash function to map a large space to a small one
  – Cannot hope to have a perfect hash function
  – Instead we settle for something that “looks random”

• The property that a hash function “looks random” can be formalized:
  – *Random* functions $\mathcal{U} \to [2^k]$ (but: too many bits)
  – *Universal* hash functions:
    \[
    \forall x_1, x_2 \in \mathcal{U}, \quad \Pr [h(x_1) = h(x_2)] \leq 2^{-k}
    \]
  – Also *strongly universal* hash functions
  – *Minwise independent* permutations (*cf.* next week)
  – *Cryptographic* hash functions

• Rich theory: see Motwani’s book or Michael Mitzemancher’s web page
Hashing In Systems

• Most common use is for hash tables such as compiler symbol table or a hash access method in a database
  – Usually worried about behavior in expectation
  – Even here, surprising theoretical results can be a big win
    * A good example is the “power of two choices” (Mitzenmacher)

• In an adversarial setting, cryptographic guarantees (read: assumptions) may be desirable or necessary
  – Digital signatures, tamper detection, capabilities, and so on

• We will look at applications from roughly the last ten years
  – Cryptographic hashes that are “perfect in practice”
  – Polynomial (Rabin) hashes that have useful algebraic properties
Cryptographic Hashing

- Theoretical ideal: any polynomial time adversary can invert hash function only with negligible probability

- Systems reality: constructing cryptographic hashes is a dark art
  - There are standard analytical tools
  - But everything rests on reasonable but unproven assumptions
  - “Strong enough” is an ever-moving target

- Old favorites were DES, MD4, and MD5, but they are no longer safe

- More recently, cryptanalysts have begun to make progress on SHA
  - Current standby is SHA-1 and its more recent cousins
  - SHA-1 maps an arbitrary length string to 160 bits
    * Typically, we assume SHA-1 is a random oracle
Polynomial Hashing

• For the details, see Rabin’s technical report (only now as PDF!)
  – Galois theory is beautiful, but takes too long to develop here
• Basic operation is reduction modulo an irreducible polynomial
  – Often simpler to just work directly in $\mathbb{Z}/2^8\mathbb{Z}$
• For an irreducible polynomial $p$ of degree $n$, compute residues of the monomial $ax^{n+1} \mod p(x)$ for every $a$ in the ground field
  – Multiplication by $x^n$ corresponds to shifting
  – Addition and subtraction of polynomials corresponds to bit-wise xor
  – Keep a table of residues, shift and subtract residue to update hash
• From a systems perspective, the issue is managing the L1 and L2 caches
Useful Properties of Polynomial Hashes

- For deg $p(x) = n$ and a string of length $m$, the probability of collision is bounded above by $nm^2/2^k$

- There is a natural, efficient representation of polynomials (bit strings)

- Prefix property: $H(A||B) = H(H(A)||B)$

- Given $H(A)$, $H(B)$, and $n = |B|:

\[
H(A||B) = x^n \cdot A(x) + B(x) \mod p(x) \\
= H(H(x^n) \cdot H(A)) + H(B)
\]

- Rolling property: if $A_i$ is the first $i$ symbols of $A$ in sequence, the natural algorithm yields $H(A_i)$ for all $i$ as an artifact
  - Can also compute the hash of all consecutive subsequences of length $k$ for fixed $k$ in one pass
Syntactic Similarity Search

• Goal: find syntactically similar files without $O(n^2)$ diffs

• Anchors: random vs. application-specific break points
  – Pitfalls: boiler-plate, *e.g.* PostScript prologues

• Rank is a function of the fraction of fingerprints in common
  – The similarity measure is not transitive

• Query: a single file or all files (clustering)
  – For single file query we can use rank-order and a threshold
  – For multi-file case, the output is a set of sets
    * Introduces a difficult user-interface problem
    * *E.g.*, how to handle small similarity sets that have significant
      intersection with larger similarity sets
Bandwidth Reduction

• Idea: use hashing to identify similar “chunks” of data in a protocol stream or cache and replace them on the wire with a reference
  – On a thin pipe, bandwidth reduction may also improve latency

• Identify redundant blocks by computing a rolling hash
  – Hash every 16-64 byte block in the protocol stream (cf. rsync)
  – A hash collision indicates a potentially redundant block
  – Reduce number of tests by selecting a random fraction (e.g. 1/2^{13})

• Parameters: window size; minimum, maximum, & expected chunk size

• Given a hash collision, how do we decide if the block is redundant?
Hash Collisions

- Given a hash collision, how do we decide if the block is redundant?
  - With a shared, synchronized cache we can test directly
  - Use SHA-1 to name blocks and assume collisions don’t happen

- Should system designers be wearing tin-foil hats?
  - It is one thing to use the hash as a hint, another to rely on it

- This is an issue we will revisit when we discuss Venti
Aside: Other Bandwidth Reduction Techniques

- Related work and common techniques:
  - Caching approaches: AFS and Coda, various peer-to-peer systems
  - Purely syntactic approaches: traditional compression, rsync
  - Optimistic approaches: Bayou, Lenses (B. Pierce), Unison, Tra
  - Semantic approaches: Sam (Rob Pike), Protium (Cliff Young et al.)

- Hashing based techniques have the advantage that they are protocol-agnostic and therefore more or less orthogonal

- Optimism often works well in practice but cost of failure high

- Application-specific techniques such as Sam’s Rasps and token-based consistency may expose more opportunities to optimize

- Static analyses such as in Lenses is promising but difficult
Consistent Hashing and Naming

• It is often convenient to name a block by its hash
  – Resulting name is a compact encoding, as in LBFS
  – Resulting name is easy to compute without a global name service
  – We can compute the location of object as a function of the name
    * Positive: objects are more or less uniformly distributed
    * Negative: objects are more or less uniformly distributed
    * Negative: even if results are good in expectation, variance can kill

• Resolving collisions can be painful if we require uniqueness
  – For some applications, an additional counter may suffice
**Consistent Hashing in Distributed Systems**

- Distributed hash tables for so-called “peer-to-peer” systems have been a hot topic in the last four years.

- Chord is a prototypical example and probably best documented:
  - Chord provides a distributed lookup service using consistent hashing.
  - Each node keeps pointers to a few nodes in power-of-two intervals.
  - Can therefore find any other node in $O(\log n)$ queries in the ring.

- Chord is also a good example of the difference between what theorists and system builders consider efficient:
  - Theorists get a warm, fuzzy feeling from $O(\log n)$.
  - System builders get a warm, fuzzy feeling from $O(1)$.

- In practice, one probably wants to take advantage of the query distribution (see, for instance, Beehive).
Consistent Hashing For Local Storage: Venti

- Background: Plan 9 and Ken’s dump file system
  - Optical juke with copy-on-write nightly dumps at 5AM
  - Interesting source of traces: snapshot for every night since ~1989
- Venti: from the Italian for 20
  - Venti itself is just a content-addressed block store
  - Similar to EMC’s Centera and others
  - Intended to be used as a service by application developers
  - Consistent hashing makes master-slave replication relatively easy
- There are a number of existing Plan 9 and Unix applications
  - vac: a tar replacement
  - fossil: a conventional fs with soft-updates and snapshots
  - Several physical backup programs, including one used by PDOS
Venti Overview

• A data stream is broken into a sequence of fixed size blocks
  – Techniques such as chunking in LBFS can be a big win

• Each block is fingerprinted and checked against the index
  – Should we retrieve blocks on a hash collision?

• Traditional metadata and indirect blocks become a stream of pointer blocks that are typed and stored just as ordinary blocks are
  – The result is a giant Merkle tree

• Blocks are packed into clumps and compressed

• Clumps are written to a sequence of arenas on disk
Why Is Venti Difficult to Implement?

- “When in doubt, introduce another level of indirection”
  - In a tradition FS, blocks are addressed by logical block number
  - Venti must translate each score to an LBN before issuing I/O

- Fragmentation and locality of reference
  - Second and subsequent copies of a file are scattered
  - May require aggressive caching, hinting, block-level duplication

- It is easy to implement content-addressable stores as append-only logs
  - It is much more complicated to permit deletion
  - Typical approach is some form of garbage collection
  - GC is an opportunity to reorganize, but makes replication harder

- Long-term reliability of disks (vs. tape), more complicated software
  - Given good disks and RAID implementation, may be a wash
Lessons Learned?
Future Work?

• We’re used to the LAN, but many users are stuck by long, thin pipe
  – Can we improve interactive applications over the WAN?
  – What can we do to avoid getting stuck behind a thin pipe as the
    volume of data explodes?
    * Content distribution, large scientific datasets, etc.

• How much can we hope to accomplish with syntactic similarity search?
  – How can system builders help support application-specific search in
    an application-agnostic way?

• Given that individuals can effectively treat the disk as an infinite
  resource, deletion is a function of policy rather than necessity
  – Can system builders design other policies to take advantage of
    available storage and still prevent users from drowning in a sea of
data