Venti

Ohad Rodeh

Introduction

- This lecture is based on: **Venti: a new approach to archival storage.** Sean Quinlan and Sean Dorward. Bell Labs, Lucent Technologies. FAST 2002 Conference on File and Storage Technologies.
- Venti was the first to introduce content-based hashing for archival storage purposes.
- This approach has become accepted practice since.

Motivation

- Archival storage is a second class citizen.
- Many systems do not allow access to previous versions of files or databases.
- Systems like AFS and WAFL do allow it, but only to a limited number of snapshots. Data is not kept in perpetuity.
- Common practice is to use an additional system, tape, for backup. But then, access to backup data is tedious.

Tape setup

- Tape systems are used as a form of second-level storage
- Typically: a tape backup system serves several client machines
- Backup software on the clients interfaces with the tape, reads the contents of the databases and file-systems and decides what to backup.
- Data is copied over the network to the tape system

Tape backup

- Restoring data from backup can be tedious and error prone
- The backup system violates the access permission of the file-system requiring a system-administrator or privileged software
- Restore operations are infrequent, so problems may go undetected
- Potential problems: tapes are mislabeled, reused, lost, drives wander out of alignment, technology becomes obsolete.

Trade-off

- There is a trade-off between backup and restore
  - Full backup is expensive but provides simple restore.
  - Normally, incremental backup is done. Complicates restore.

Main observations

- The growth in capacity of disk storage allows all data to be kept on disk, online.
- Use a write-one policy. Never erase data.
- Obviously, some data is too large too be kept, but once it is decided to keep it, it is never erased.
- Eliminates the tedious task of deciding what to erase.
- Simplifies the system because it does not need to overwrite nor delete data.

The Venti archival server

- Venti is a block-level network storage system.
- It is not a database or a file-system. It provides a block-based back-end for storage applications.
- Venti exports a simple interface: read/write variable sized blocks of data.

Unique hashes

- Venti identifies data blocks by a hash of their contents.
- By using a collision resistant hash function with a sufficiently large output, it is possible to consider the hash of a data block as unique.
- The hash is the *fingerprint* of the block
- The hash is then used as the *address* of the block for read/write operations
- This approach results in a storage system with some interesting properties

Hashes as addresses

- As blocks are addressed by their hash, a block cannot be modified without changing its address
- Intrinsically write-once behavior
- In most other storage systems the address of a block never changes

Backup behavior with Venti

- Multiple writes of the same data can be coalesced and do not require additional storage space.
- This simplifies backup behavior
- A client can perform a full-backup, Venti will eliminate redundancy between the new data and old data. No space will be wasted by moving from incremental to full backup.
- Furthermore, data from different applications can also be eliminated.

Hashes as addresses II

- The hash function can be viewed as generating a universal name space for data blocks
- Without cooperating or coordinating multiple clients can share the name-space of a Venti server
- The low-level block interface places few restrictions on applications
- Any application that uses a disk can use Venti
- Traditional backup systems require more control. For example, they need to crawl over the database or file-system and differentiate new data from old-data in order to perform incremental backup.

Hashes as addresses III

- The hash of a block is also used for integrity checking
- Since the contents of a block are immutable, the problem of data coherency is greatly reduced.
- A cache or a mirror cannot contain stale data

Choice of hash function

- Venti requires a hash function that generates a unique fingerprint for every block
- In practice, this is done using a cryptographic hash
- Venti uses Sha1 which has an output of 160bits
- Probability of a collision:
  1. n is the number of blocks
  2. b is the number of bits in the hash:

     $p \leq \frac{n(n-1)}{2} \times \frac{1}{2^b}$

Likelihood of a collision

- In a system with 10^{18} bytes of data stored as 8KB blocks the probability of a collision is about 10^{-20}
- This is sufficiently unlikely to be ignored
- In the future, larger hashes can be used.

Using Venti by an application

- Venti poses a unique challenge to an application
- Writing is performed by sending data blocks to the Venti server
- In order to read a block of data, its fingerprint must be provided by the application.
- This requires the application to store block fingerprints

Tree of fingerprints

- Store the tree of fingerprints on the server.

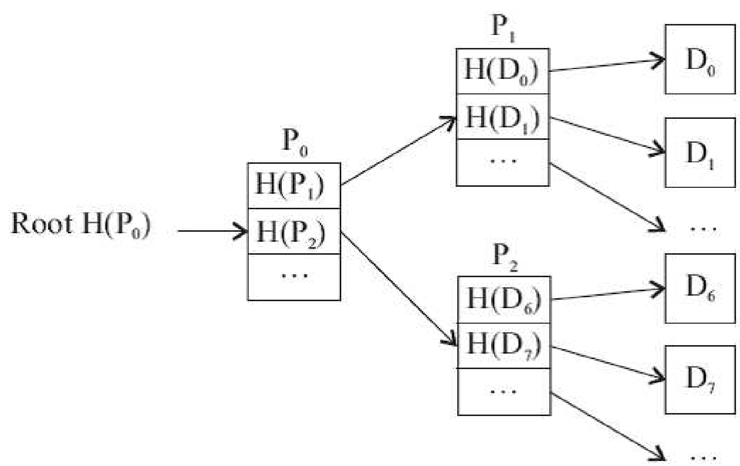

Figure 1. A tree structure for storing a linear sequence of blocks

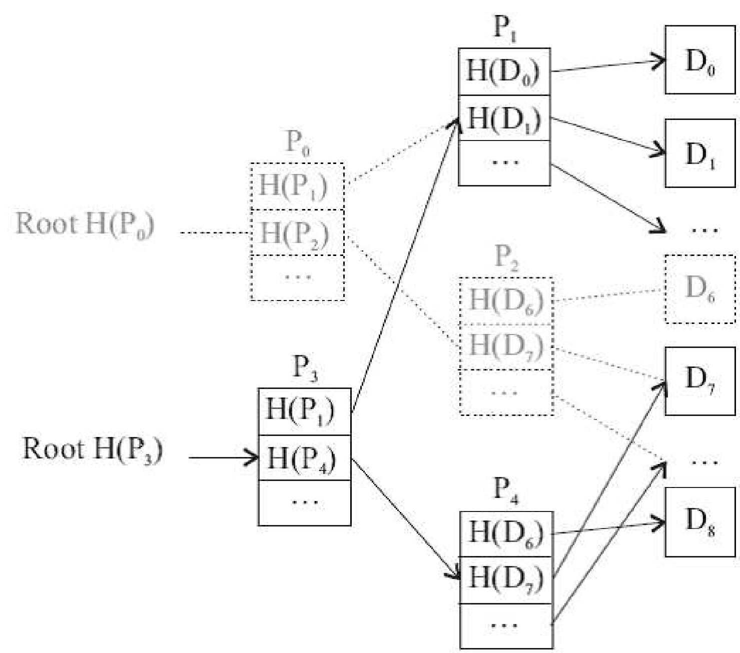

Figure 2. Build a new version of the tree.

Vac

- *Vac* is like tar or zip, an application that stores many files and directories as one object.
- The contents of files are stored as a tree of blocks
- The root fingerprint is stored at the file (*.vac) specified by the user
- The fingerprint is stored in ascii format plus a header for a total of 45 bytes
- This makes it look as if the entire archive takes up 45 byte on disk

Vac II

- Vac stores each file as a separate tree of blocks
- This ensures that duplicate copies of a file will be coalesced on the server
- If multiple users *vac* the same data, it will be stored on the server exactly once.
- Repeatedly vac-ing the same directory will not use up more storage
- Even if the directory changes, only the changes take up additional space

The plan-9 file-system

- Plan-9 is a Unix-like operating system from Bell labs
- It can be downloaded from the bell-labs site
- It has a file system that supports snapshots
- Previously, the plan9-FS was stored on a combination of magnetic disks and write-once optical jukebox

The plan-9 file-system II

- The jukebox provides permanent storage
- The disks act as a writeable-cache for the jukebox
- The cache provides faster access and also accumulates changes between snapshots
- When a snapshot is taken, all modified blocks are written to permanent storage

The Plan-9 file-system III

- The cache can be smaller than the active file-system
- However, accesses that miss the cache are significantly slower since changing platters in the jukebox takes seconds
- This performance penalty makes certain operations on backup storage prohibitively expensive
- Also, when the cache is reinitialized due to corruption, the file-server spends several days filling the cache before performance returns to normal

Plan9-FS based on Venti

- The new version of Plan9-FS uses Venti as its back-end instead of an optical jukebox
- This simplifies things because Venti has the same latency as a disk

Implementation

- An append-only log for data blocks
- An index that maps fingerprints to locations in the log

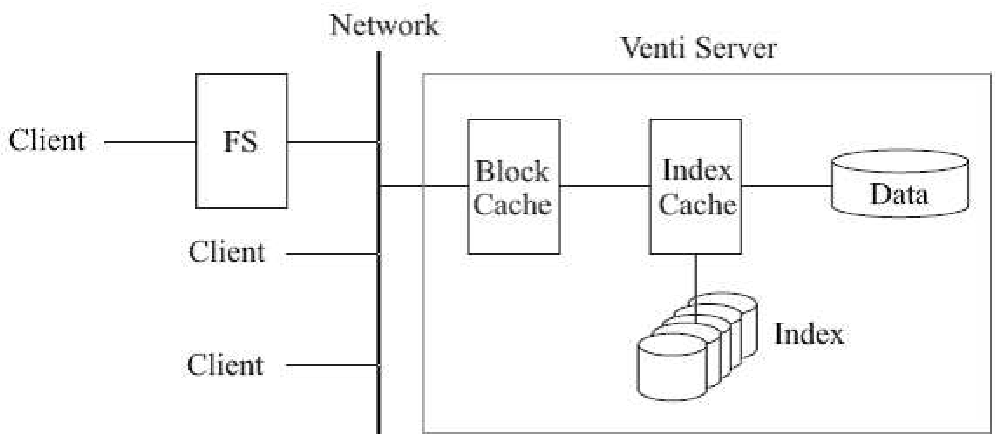

Figure 3. A block diagram of the Venti prototype.

The log

- Since Venti is intended for archival storage, robustness is particularly important
- The log is placed on a RAID array to protect against disk errors
- The log is append only, data is never overwritten nor erased
- The log is separated into arenas
- Data is compressed before being inserted into an arena

The log II

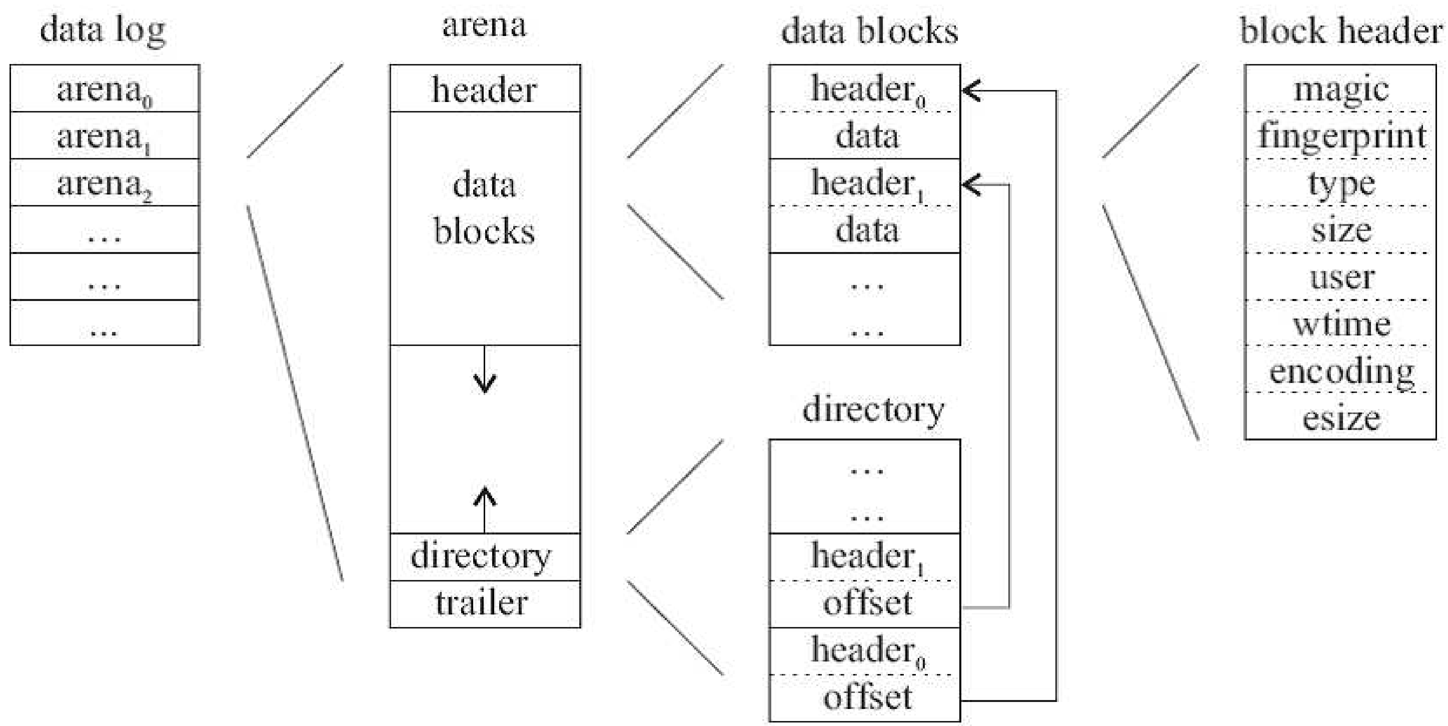

Figure 4. The format of the data log.

The index

- The index is implemented by a disk resident hash-table
- The index is divided into fixed-sized buckets
- Each bucket is stored as a single disk-block
- Each bucket contains the index-map for a small section of the fingerprint space
- A hash function is used to map fingerprints into buckets in a relatively uniform fashion.
- Binary search is used inside a bucket
- This structure provides an almost always one disk access per lookup.

The index II

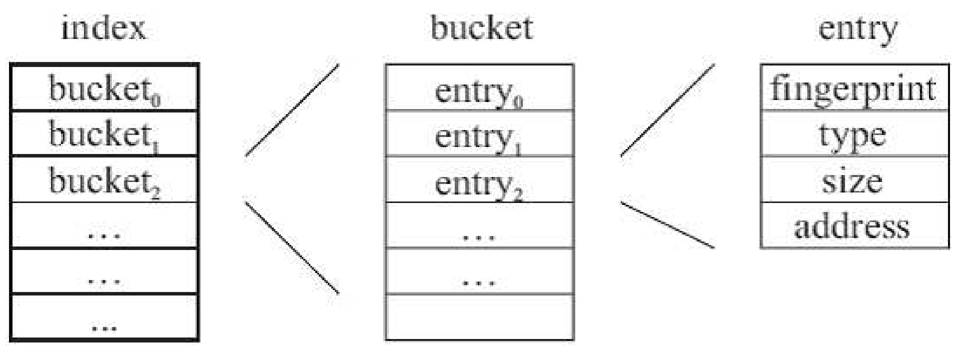

Figure 5. Format of the index.

Performance issues

- The main disadvantage of Venti compared to a standard block-based system is the need to go through the index
- Three techniques are used to offset this disadvantage: caching, striping, and write buffering.

Caching

- One cache for the index, another for data blocks.
- First, the data cache is checked. Second, the index cache.
- Caching, however, does not help writing new data much.
- First, Venti needs to check if the block already exists. Since the block is new, it is obviously not in cache.
- Since the fingerprint is essentially random, it will most likely miss the index cache.
- Therefore, the write performance will be limited to the random IO performance of the index disk.

Hardware

- The prototype Venti server is implemented with
  1. Plan 9 operating system
  2. 10000 lines of C code
  3. Dual 550Mhz Pentium III
  4. 2GB of memory
  5. 100Mbit/sec Ethernet
  6. The data log is stored on 500GB MaxTronic IDE RAID-5 array
  7. The index resides on a string of 8 Seagate Cheetah 18XL 9GB SCSI disks

Base performance

- The main problem occurs when performing un-cached reads

Table 1. The performance of read and write operations in Mbytes/s for 8 Kbyte blocks

	Sequential Reads	Random Reads	Virgin Writes	Duplicate Writes
Uncached	0.9	0.4	3.7	5.6
Index Cache	4.2	0.7	-	6.2
Block Cache	6.8	-	-	6.5
Raw Raid	14.8	1.0	12.4	12.4

Historical data

- Two file servers: Bootes and Emelie
- Boots: 1990-1997, block-size 6KB
- Emelie: 1998-2001, block-size 16KB
- Total of 522 user accounts, 50-100 active at any one time
- Large data sets: astronomical data, chess end games, etc.

File servers

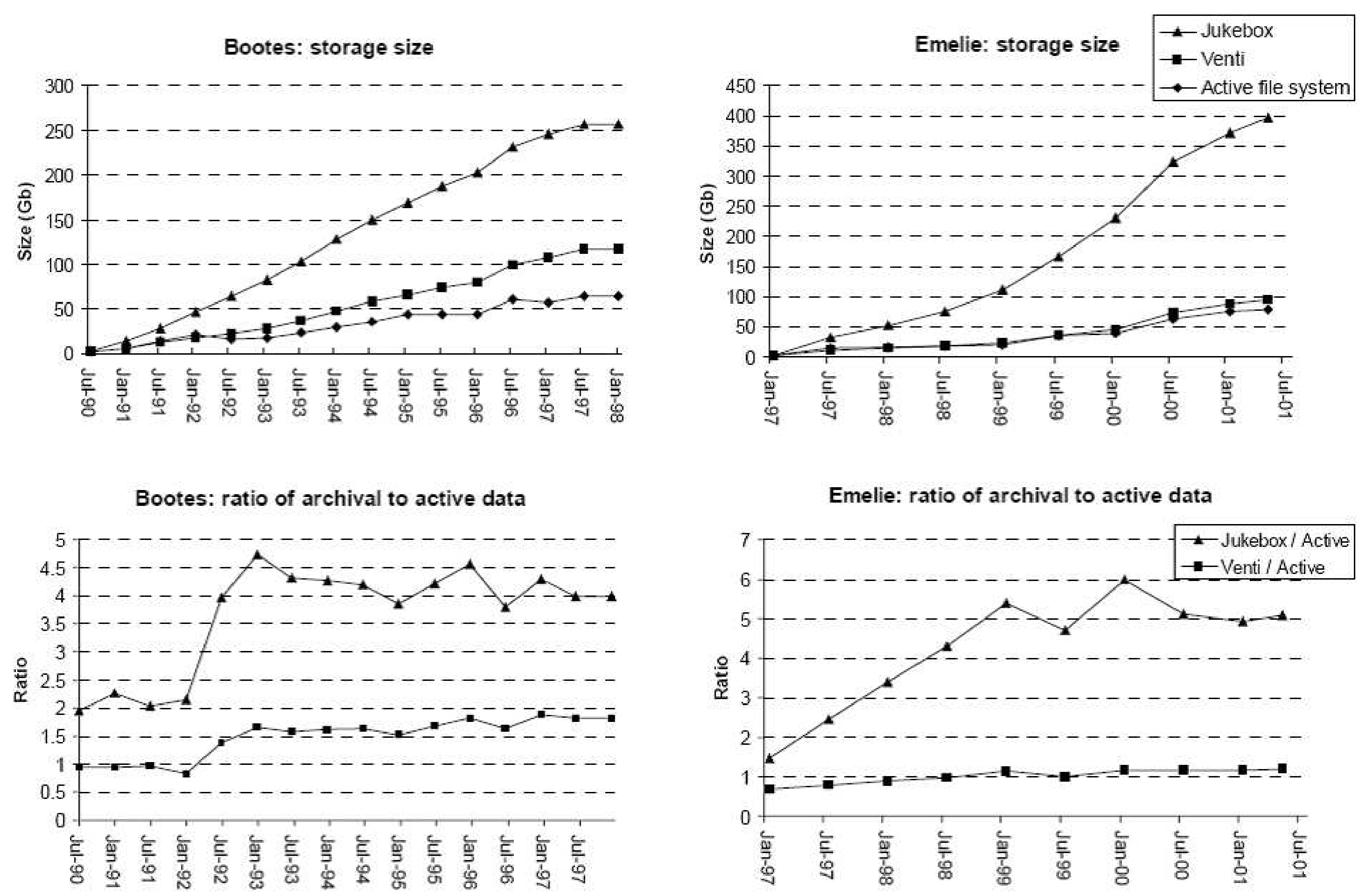

Figure 6. Graphs of the various sizes of two Plan 9 file servers.

De-duplication

Table 2. The percentage reduction in the size of data stored on Venti.

	bootes	**emelie**
Elimination of duplicates	27.8%	31.3%
Elimination of fragments	10.2%	25.4%
Data Compression	33.8%	54.1%
Total Reduction	59.7%	76.5%

Reliability and Recovery

- Part of Venti's charter is to build a reliable permanent store
- Special tools were built to check for integrity and recover from corruption
- Examples:
  1. Verifying the structure of an arena
  2. Checking that there is an index entry for each data block
- These tools directly access the storage and are executed on the server

Summary

- Venti introduced the concept of content-addressable storage
- Now a fully established concept
- EMC sells the Centera product