BorgFS File System Metadata Index Search

Stephen P. Morgan

Masood Mortazavi

Huawei Technologies Co., Ltd.

Overview

- Background
- Development Choices
- Detailed Design
- Evaluation
- Future Work

Background (1)

- ❒ BorgFS is a Scale-Out File System from Huawei
- ❒ Provides POSIX Interface
- ❒ Implements Deduplication / Erasure Coding
- ❒ Uses Low-Cost Processors in Storage Nodes
- ❒ Has Much Lower $/IOPS Than Industry Leaders
- ❒ Scales to a Billion Files

Background (2)

- With a Billion Files, Easy to Lose Track by Name
- Hierarchical Naming is Helpful—Up to a Point
- Pathname and Metadata-based Search Simplifies Finding Lost Files
- Use Filesystem and Custom Metadata
    - size, date, partial path, file format, user tag

Query Examples

- path=/borgfs/pfs & base=Makefile & size<1000
- cdate>2014Aug18 & cdate<2014Aug19 & base=memo & path=/home/smorgan/mydocs
- base=alice & format=jpeg & fstop=1.4

Prior Work

System	Source	Year	Technology	Custom Metadata	Replace Directory Layout
Spyglass	NetApp & UCSC	2009	KD-Tree & BF	No	No
Magellan	UCSC	2009	KD-Tree & BF	No	Yes
LazyBase	HP & CMU	2014	RDBMS	No?	No
InfoExplorer OceanStor 9000	Huawei	2014	RDBMS & K-V Store	Yes	No
BorgFS	Huawei	2014	K-V Store (LSM) & BF	Yes	No

K-D Tree

- Space-Partitioning Data Structure for Organizing Points in K-Dimensional Space
- Useful for Multidimensional Search
- Difficult to Support Custom Attributes

Bloom Filter

- Probabilistic Data Structure Used to Determine Whether Element is Member of Set
- False Positives are Possible, But Not False Negatives
  - Thus Either Probably in Set or Definitely Not
- Useful in Partitioning Search Space

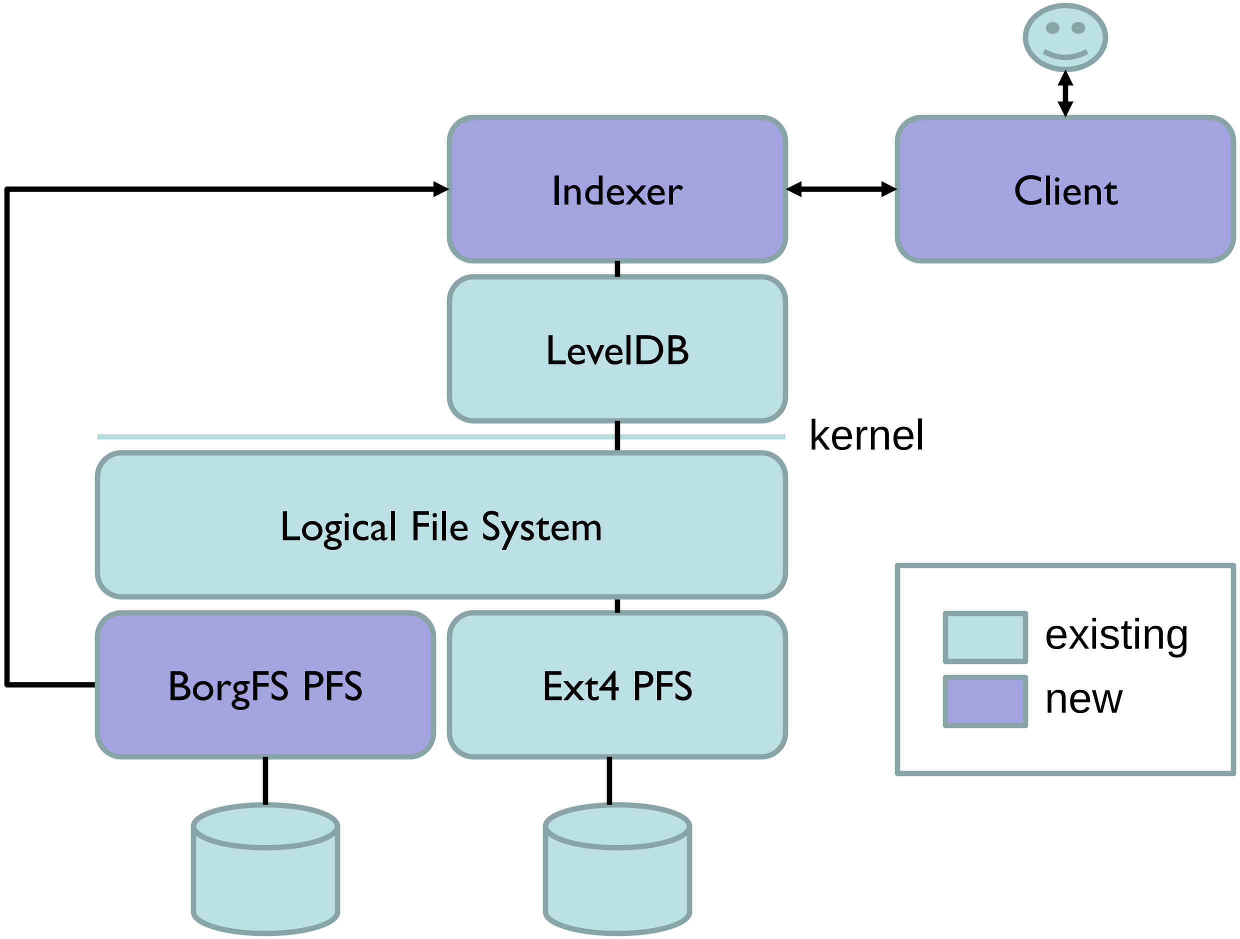
Overview

- ❐ Background
- ❐ Development Choices
- ❐ Detailed Design
- ❐ Evaluation
- ❐ Future Work

Development Constraints

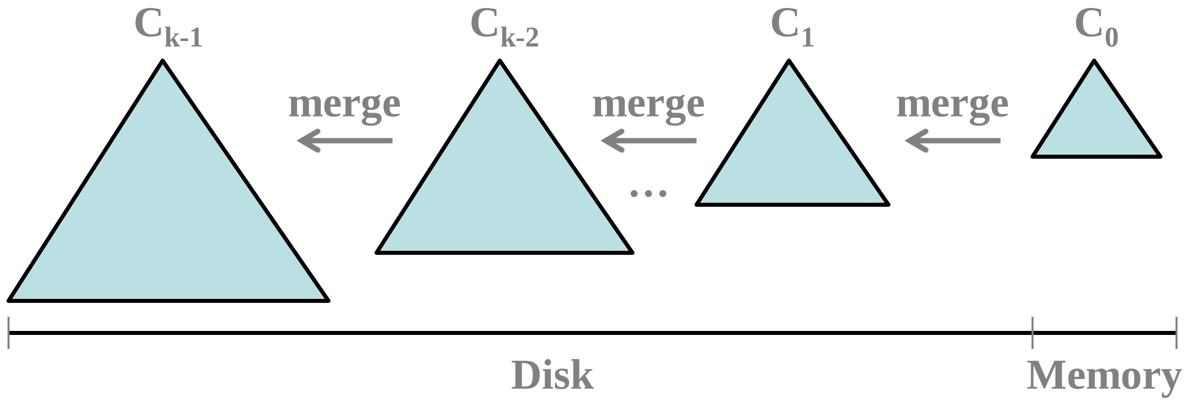
- Index had to Support one Billion Files
- Had to be Reasonably Fast for Typical Queries
- Indexer had to Run on a Single, Modest Server
- Had to Integrate with BorgFS Physical FS
- Had to Ingest Filesystem Changes in Real Time
- Had to be Developed Quickly and at Low Cost

Overall Architecture



LevelDB

- Open-source Key-Value Store from Google
- Implements Log-structured Merge Tree



Why LevelDB?

- Much Simpler than RDBMS
- Able to Support Custom Attributes
  - Unlike K-D Tree
- High Performance for Updates and Queries
- Cascaded Levels of Trees Data Structure
- Open Source Available

Overview

- ❐ Background
- ❐ Development Choices
- ❐ Detailed Design
- ❐ Evaluation
- ❐ Future Work

Database Tables (1)

- The database includes the following tables: base name, full path name, type, size, access time, modification time, change time, number of links, user identifier, group identifier, and permissions
- These tables map file system metadata items to inode number and device number
- To support hard links, another table is maintained for the inverted relationship between an inode and its potentially multiple names

Database Tables (2)

- Another table maintains the full filesystem metadata in one place
- An optional table maintains custom metadata

Schema (1)

- A file named, “/borgfs/a/b/c/data.c” with inode number 12 and device number 2048 has the PATH table key of “/borgfs/a/b/c/data.c:0000002048:0000000012” along with an empty value
- If the file has one link, it has an entry with the key of “0000000001:0000002048:0000000012” added to the LINKS table, along with an empty value

Schema (2)

- ❐ LevelDB has prefix search and Seek() to first matching prefix
- ❐ Performing a prefix search on part of key before first ":", yields device number and inode number that identifies file. For PATH, will be unique; for LINKS, there will be many
- ❐ Use **strtok**() to separate attributes using ":"
- ❐ Allows use of K-V store as relational database

Schema (3)

- The inverted pathname table INVP would contain the key “0000002048:0000000012:/borgfs/a/b/c/data.c” along with an empty value
- Another entry (for the same file) might be the key “0000002048:0000000012:/borgfs/data.c” along with an empty value if /borgfs/data.c is a hard link to /borgfs/a/b/c/data.c

Schema (4)

- To find the pathnames for the inode 12 on device 2048, Seek() to "0000002048:0000000012:" then iterate through keys in INVP table. Use **strtok**() to extract pathname from key using ":" as the separator
- Yields inverted inode-to-pathname list, e.g., "0000002048:0000000012:/borgfs/a/b/c/data.c" and "0000002048:0000000012:/borgfs/data.c", or
  - /borgfs/a/b/c/data.c and /borgfs/data.c

Filesystem Metadata Maintained in MAIN

- The file with device number 2048 and inode number 12 has the key “0000002048:0000000012” along with the value “R:0644:0000000001:0000000100:0000000101:0000065536:1000000001:1000000002:1000000003” in the table MAIN if it is a regular file with permissions 0644 (octal), has one link, is owned by userid 100 and group id 101, contains 65,536 bytes, has an access time of 1000000001, a change time of 1000000002, and a modification time of 1000000003 seconds

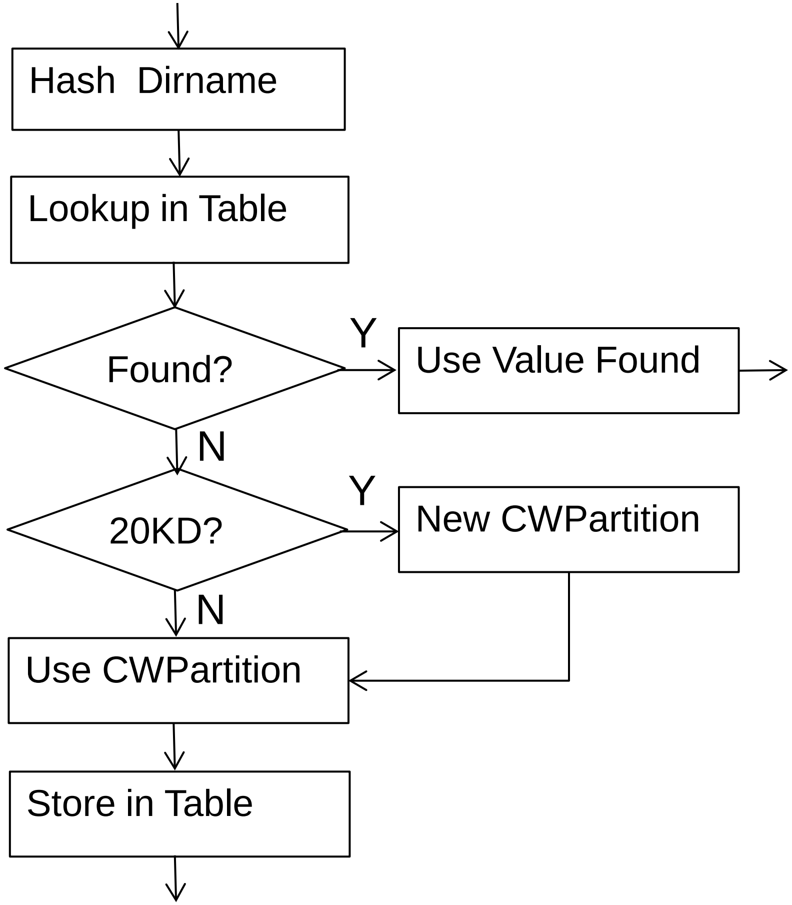
Custom Metadata Maintained in CUSTOM

- ❒ Add a CUSTOM table with custom (i.e., non-filesystem) attributes and values
- ❒ For example, to add “format” tag (metadata) with “mpeg4” value, add entry to CUSTOM table with the key (along with an empty value) “format:mpeg4:0000002048:0000000012”
- ❒ To search for files with format=mpeg4, use prefix search. Use **strtok**() to extract device:ino from key. Results in file with device of 2048 and inode of 12.

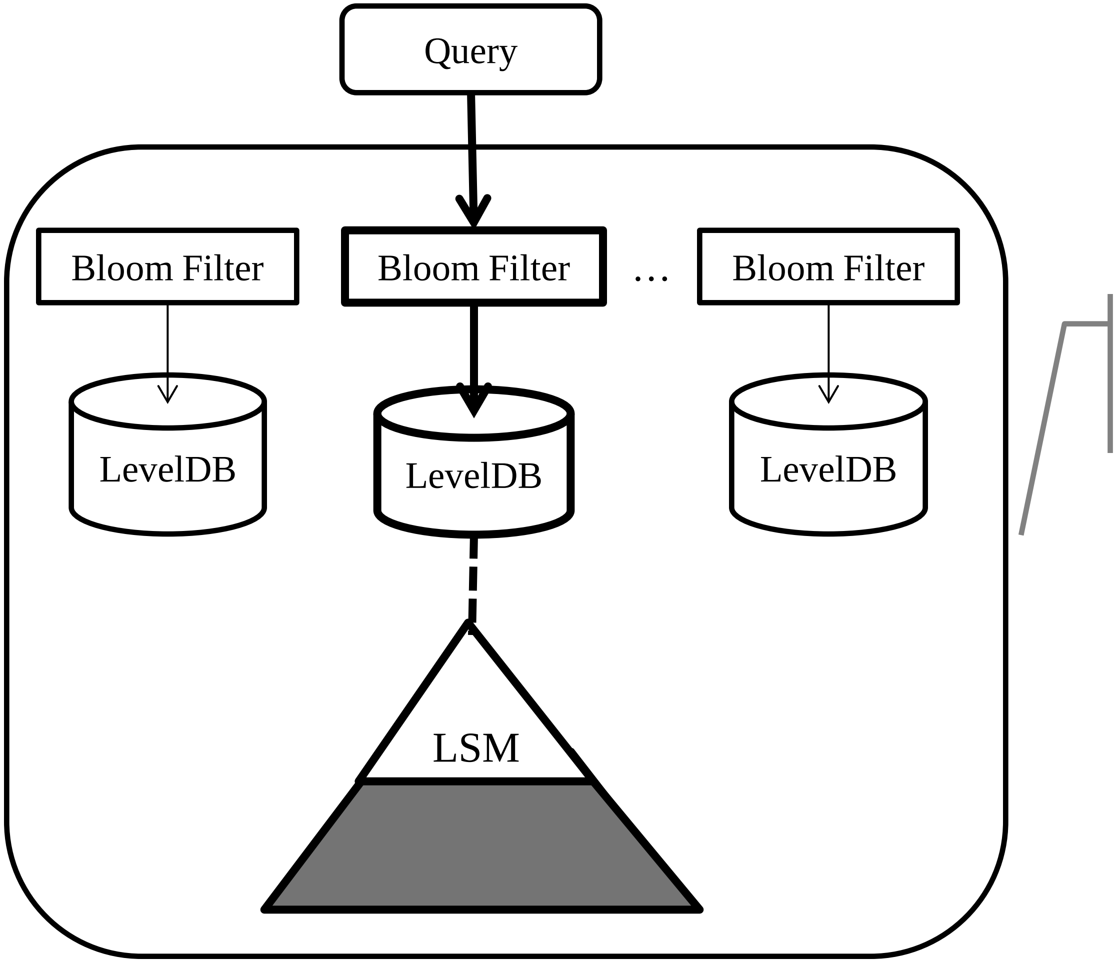
Partitioned System

- Divided System Into Up To 1,000 Partitions by Hashing on Directory Name at File Create Time
- Tried to Put Up to 20,000 Directories Together
- Each Partition Had Its Own LevelDB Database
- Used Bloom Filters to Decide Which Partitions to Search When Running Queries
  - Typical Query Included Partial Path Name
- Bloom Filter had 32K Bits and 4 Hash Functions

Hash Table


Hash Dirname
Lookup in Table
Found?
Y
Use Value Found
N
20KD?
Y
New CWPartition
N
Use CWPartition
Store in Table

Query Processing



1. Select Bloom Filters
2. Query LevelDB Stores
3. Scan LSM Trees
4. Join and Merge Results

Overview

- ❒ Background
- ❒ Development Choices
- ❒ Detailed Design
- ❒ Evaluation
- ❒ Future Work

Example Queries (1)

- path=/borgfs/006/6/6 & base=random.c
  - Searched one copy of Linux kernel
  - Retrieved 2 files in 0.289 seconds

- path=/borgfs/000/6/9 & base=Makefile
  - Searched one copy of Linux kernel
  - Retrieved 1,526 files in 2.145 seconds

Example Queries (2)

- path=/borgfs/006/6 & links>1
  - Searched ten copies of Linux kernel
  - Retrieved 41,051 files in 14.272 seconds
- path=/borgfs/006 & base=random.c
  - Searched 100 copies of Linux kernel
  - Retrieved 200 files in 19.346 seconds

Example Queries (3)

- /path=/borgfs & base=random.c
  - Searched 1,200 copies of Linux kernel
  - Retrieved 2,400 files in 6m56.561 seconds

Overview

- ❒ Background
- ❒ Development Choices
- ❒ Detailed Design
- ❒ Evaluation
- ❒ Future Work

Future Work

- Switch from LevelDB to RocksDB
  - Est. Query Time About 3X Faster
- Support Fuzzy Search
  - Start with Wildcards
- Support High Availability / Failover / Restart

Thank You!