LevelDB Introduction

An Key-Value Store Example
Projects Using LevelDB

- Chrome
- BigTable
- Riak
- Kyoto Tycoon
LevelDB

• “LevelDB is an open source on-disk key-value store written by Google fellows Jeffrey Dean and Sanjay Ghemawat.” – Wikipedia

• “LevelDB is a light-weight, single-purpose library for persistence with bindings to many platforms.” – leveldb.org
API

• Get, Put, Delete, Iterator (Range Query).
Key-Value Data Structures

• Hash table, Binary Tree, B⁺-Tree

"when writes are slow, defer them and do them in batches” *

*Dennis G. Severance and Guy M. Lohman. 1976.*
Log-structured Merge (LSM) Tree

Two Component LSM-Tree

Figure 2.1. Schematic picture of an LSM-tree of two components
K+1 Components LSM-Tree

**Figure 3.1.** An LSM-tree of K+1 components
Rolling Merge

Figure 2.2. Conceptual picture of rolling merge steps, with result written back to disk
From LSM-Tree to LevelDB

LevelDB Data Structures

- Log file
- Memtable
- Immutable Memtable
- SSTable (file)
- Manifest file
Archival Storage
Outline

• Archival Storage
  - archival
  - backup vs archival
• Long-term data retention
  - architecture and technologies
  - cloud for archival
  - Self-contained Information Retention Format
What is archival storage?

• In computers, archival storage is **storage** for data that may not be actively needed but is kept for possible **future use** or for **record-keeping** purposes.

• Archival storage is often provided using the same system as that used for backup storage. Typically, archival and backup storage can be retrieved using a restore process [1].
The Need for Digital Preservation

- Regulatory compliance and legal issues
  - Sarbanes-Oxley, HIPAA, FRCP, intellectual property litigation
- Emerging web services and applications
  - Email, photo sharing, web site archives, social networks, blogs
- Many other fixed-content repositories
  - Scientific data, intelligence, libraries, movies, music

Scientific and Cultural
Satellite data is kept for ever
We would like to keep digital art for ever

Healthcare
X-rays are often stored for periods of 75 years
Records of minors are needed until 20 to 43 years of age

M&E
Film Masters. Out takes. Related artifacts (e.g., games). 100 Years or more

SIRF: Self-contained Information Retention Format
Approved SNIA Tutorial © 2015 Storage Networking Industry Association. All Rights Reserved.
An Archival Storage System

• A high-end computing environment includes a 132-petabyte tape storage system that allows science and engineering users to archive and retrieve important results quickly, reliably, and securely (NASA)

• **44 PB** current unique data stored

• SGI
Backups and Archives

• Backups are for recovery

• Archives are for discovery and preservation
Storage Perspective: archival application

• Data archiving is the process of moving data that is no longer actively used to a separate data storage device for long-term retention.

• Most are write once, but if needed, it is crucial
<table>
<thead>
<tr>
<th>Issue</th>
<th>Backup</th>
<th>Archiving</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is it?</td>
<td>Protection for system and data &quot;live state&quot;</td>
<td>Records of inactive document &quot;steady state&quot;</td>
</tr>
<tr>
<td>Why use it?</td>
<td>Recovery&quot; restore business operations after data loss, interruption, or disaster</td>
<td>Discovery: produce evidence to meet legal, regulatory, and policy obligations</td>
</tr>
<tr>
<td>Who wants it?</td>
<td>Business stakeholders—CEO, shareholders, your boss</td>
<td>Public stakeholders—courts, regulators</td>
</tr>
<tr>
<td>What's in it?</td>
<td>Images in full operational context</td>
<td>Individual objects, especially email</td>
</tr>
<tr>
<td>How many are there?</td>
<td>Many: original left in place plus multiple point-in-time copies</td>
<td>One: single global instance – originals replaced by links, or removed altogether from primary storage</td>
</tr>
</tbody>
</table>
Backup and disaster recovery requirements

• High media capacity
• High-performance read/write streaming
• Low storage cost per GB
Archive requirements

• Data authenticity
• Extended media longevity
• High-performance random read access
• Low total cost of ownership
Long Term Data Retention – 5 Key Considerations

1. Business and Regulatory Requirements Demand a Long-term Plan
2. Manage and Contain Your Total Cost of Ownership (TCO)
3. Encrypt Your Data for Secure Long-term Retention
4. Weigh the Environmental Impacts and Minimize Power and Cooling Costs
5. Simplify Management of the Entire Solution
Disk scrubbing

• Drives are periodically accessed to detect drive failure. By scrubbing all of the data stored on all of the disks, we can detect block failures and compensate for them by rebuilding the affected blocks.
The two-tiered data retention

The two-tiered architecture enables administrators to deploy a short-term active tier for fast ingest of backup data, and a retention tier for cost-effective long-term backup retention [7] (Data Domain).
The Emergence of a New Architecture for Long-term Data Retention

• By taking advantage of the tape layer, use cases like archiving, long-term retention and tiered storage (where 70+% of the data is stale) can live on a low-cost storage medium like tape.

• By leveraging Flash/SSD, each use case doesn’t suffer the typical tape performance barriers.
File Systems

Files
Directories
File system implementation
Example file systems
Long-term Information Storage

1. Must store large amounts of data

2. Information stored must survive the termination of the process using it

3. Multiple processes must be able to access the information concurrently
# File Naming

<table>
<thead>
<tr>
<th>Extension</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.bak</td>
<td>Backup file</td>
</tr>
<tr>
<td>file.c</td>
<td>C source program</td>
</tr>
<tr>
<td>file.gif</td>
<td>Compuserve Graphical Interchange Format image</td>
</tr>
<tr>
<td>file.hlp</td>
<td>Help file</td>
</tr>
<tr>
<td>file.html</td>
<td>World Wide Web HyperText Markup Language document</td>
</tr>
<tr>
<td>file.jpg</td>
<td>Still picture encoded with the JPEG standard</td>
</tr>
<tr>
<td>file.mp3</td>
<td>Music encoded in MPEG layer 3 audio format</td>
</tr>
<tr>
<td>file.mpg</td>
<td>Movie encoded with the MPEG standard</td>
</tr>
<tr>
<td>file.o</td>
<td>Object file (compiler output, not yet linked)</td>
</tr>
<tr>
<td>file.pdf</td>
<td>Portable Document Format file</td>
</tr>
<tr>
<td>file.ps</td>
<td>PostScript file</td>
</tr>
<tr>
<td>file.tex</td>
<td>Input for the TEX formatting program</td>
</tr>
<tr>
<td>file.txt</td>
<td>General text file</td>
</tr>
<tr>
<td>file.zip</td>
<td>Compressed archive</td>
</tr>
</tbody>
</table>

Typical file extensions.
File Structure

- Three kinds of files
  - byte sequence
  - record sequence
  - tree
File Types

(a) An executable file   (b) An archive
File Access

• Sequential access
  • read all bytes/records from the beginning
  • cannot jump around, could rewind or back up
  • convenient when medium was mag tape

• Random access
  • bytes/records read in any order
  • essential for data base systems
  • read can be …
    • move file marker (seek), then read or …
    • read and then move file marker
# File Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protection</td>
<td>Who can access the file and in what way</td>
</tr>
<tr>
<td>Password</td>
<td>Password needed to access the file</td>
</tr>
<tr>
<td>Creator</td>
<td>ID of the person who created the file</td>
</tr>
<tr>
<td>Owner</td>
<td>Current owner</td>
</tr>
<tr>
<td>Read-only flag</td>
<td>0 for read/write; 1 for read only</td>
</tr>
<tr>
<td>Hidden flag</td>
<td>0 for normal; 1 for do not display in listings</td>
</tr>
<tr>
<td>System flag</td>
<td>0 for normal files; 1 for system file</td>
</tr>
<tr>
<td>Archive flag</td>
<td>0 for has been backed up; 1 for needs to be backed up</td>
</tr>
<tr>
<td>ASCII/binary flag</td>
<td>0 for ASCII file; 1 for binary file</td>
</tr>
<tr>
<td>Random access flag</td>
<td>0 for sequential access only; 1 for random access</td>
</tr>
<tr>
<td>Temporary flag</td>
<td>0 for normal; 1 for delete file on process exit</td>
</tr>
<tr>
<td>Lock flags</td>
<td>0 for unlocked; nonzero for locked</td>
</tr>
<tr>
<td>Record length</td>
<td>Number of bytes in a record</td>
</tr>
<tr>
<td>Key position</td>
<td>Offset of the key within each record</td>
</tr>
<tr>
<td>Key length</td>
<td>Number of bytes in the key field</td>
</tr>
<tr>
<td>Creation time</td>
<td>Date and time the file was created</td>
</tr>
<tr>
<td>Time of last access</td>
<td>Date and time the file was last accessed</td>
</tr>
<tr>
<td>Time of last change</td>
<td>Date and time the file has last changed</td>
</tr>
<tr>
<td>Current size</td>
<td>Number of bytes in the file</td>
</tr>
<tr>
<td>Maximum size</td>
<td>Number of bytes the file may grow to</td>
</tr>
</tbody>
</table>

Possible file attributes
File Operations

1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename
An Example Program Using File System Calls (1/2)

/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);  /* ANSI prototype */

#define BUF_SIZE 4096
#define OUTPUT_MODE 0700

/* use a buffer size of 4096 bytes */
/* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);  /* syntax error if argc is not 3 */
/* Open the input file and create the output file */
in_fd = open(argv[1], O_RDONLY);    /* open the source file */
if (in_fd < 0) exit(2);             /* if it cannot be opened, exit */
out_fd = creat(argv[2], OUTPUT_MODE); /* create the destination file */
if (out_fd < 0) exit(3);            /* if it cannot be created, exit */

/* Copy loop */
while (TRUE) {
    rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
    if (rd_count <= 0) break;                /* if end of file or error, exit loop */
    wt_count = write(out_fd, buffer, rd_count); /* write data */
    if (wt_count <= 0) exit(4);             /* wt_count <= 0 is an error */
}

/* Close the files */
close(in_fd);
close(out_fd);
if (rd_count == 0) /* no error on last read */
    exit(0);
else
    exit(5);                             /* error on last read */
Memory-Mapped Files

(a) Segmented process before mapping files into its address space

(b) Process after mapping existing file *abc* into one segment creating new segment for *xyz*
Directories
Single-Level Directory Systems

• A single level directory system
  • contains 4 files
  • owned by 3 different people, A, B, and C
Cloud Storage and Big Data

- OpenStack
- VM vs. Container
- Durability, Reliability and Availability
- Private vs. Public Cloud
Parallel File Systems and IO
Workload Characterization
Why Is This Important?

- **Workload Characterization**
  - Key to *performance analysis* of storage subsystems.
  - Key to the *implementation of simulators*, as captured/synthesized workloads are key inputs.

- **Key Issues**
  - Lack of *widely available tool sets* to capture file system level workloads for parallel file systems
  - Lack of *methods to characterize* parallel workloads (for parallel file systems)
  - Lack of *methods to synthesize* workloads accurately at *all levels* (Block, File, etc)
  - Understanding of how existing workloads *scale* in the *exascale* regime is lacking
Goals and Objectives

• *A detailed understanding and survey* of existing methods in file system tracing, trace replaying, visualization, synthetic workload generators at the file system input levels, and existing mathematical models

• *Tools, techniques and methods* to analyze parallel file system input traces *(require to know more about OS, meta-data server, and applications)*

• Models to *characterize* the above workloads traces (Using statistical and analytical methods)

• *Synthetic workload generation* at the parallel file system input level – which will be used as inputs to the simulator.

• *Understanding of the interactions of workloads* at the file system level and making the file system aware of the workloads
Block-Level Workload Characterization

Storage system performance cannot be determined by the system alone.

- \( P = f(S, W) \)
- Improving system for all possible workload space is difficult.
- If we know the real workload space we can improve performance more efficiently.

Possible Workload Space

Real Workload Space
Framework of I/O Workload Characterization

Original trace → Workload characterization → Workload Parameters → Parameter adjustment → Adjusted Parameters → Workload generation → Synthetic trace

Comparison 1: Arrival pattern, File/Data access pattern in the form of parameters → Changes to applications and/or system (either host or storage) → Comparison 3

Replayed trace → Replay by workload replayer → Replay on same/different storage system → Comparison 2

Action: Arrival pattern, File/Data access pattern in the form of parameters
Output: Changes to applications and/or system (either host or storage)
Tiered Storage Research

File Users

I/O Storage Subsystem

Units of RAID5 Sets

Active-Tier

Passive-Tier

Tape/Library

Tier 1

Tier 2

Tier 3

A2P–Migrate

P2T–Migrate

Reads
Data Migration, Duplication, and Deduplication

- Tiered Storage Management
- When a file is accessed, we may want to move related data level up to a faster storage provisioning potential near future access requests
- Duplication level optimal for a long-term storage
- Dedup algorithm and how to preserve it long-term (need to make sure we know how to get the data back)
- How to find the right balance between duplication and dedup? How do we validate that data is stored the way we think it is?
- Imperfect dedup may be what we are looking for. However, what do we do if we want to have different levels of backup for different data.
DNA-Storage
Background

DNA Basics

What Does DNA Look Like?

https://www.genome.gov/Pages/Education/Modules/BasicsPresentation.pdf
Background

PCR: polymerase chain reaction

• PCR: a method for exponentially amplifying the concentration of selected sequences of DNA within a pool.

• Primers: The DNA sequencing primers are short synthetic strands that define the beginning and end of the region to be amplified.
Polymerase chain reaction - PCR

1. **Denaturation** at 94-96°C
2. **Annealing** at ~68°C
3. **Elongation** at ca. 72°C

https://en.wikipedia.org/wiki/Polymerase_chain_reaction
Background

DNA Synthesis

- Arbitrary single-strand DNA sequences can be synthesized chemically, nucleotide by nucleotide.
- Synthesizing error limits the size of the oligonucleotides (< 200 nucleotides).
  - truncated byproducts
- Parallel synthesize: $10^5$ different oligonucleotides.
Background

DNA sequencing

• The DNA strand of interest serves as a template for PCR.
• Fluorescent nucleotides are used during this synthesis process.
• Read out the complement sequence optically.
  • Read error. (~1%)
A DNA Storage System

• Very **dense** and **durable** archival storage with access times of many hours to days.
• DNA synthesis and sequencing can be made arbitrarily **parallel**, making the necessary read and write **bandwidths** attainable.
Overview

• **basic unit**: DNA strand that is roughly 100-200 nucleotides long, capable of storing 50-100 bits total.

• **data object**: maps to a very large number of DNA strands.

• The DNA strands will be stored in **pools**
  
  • stochastic spatial organization
  
  • structured addressing: impossible
  
  • address: embedded into the data stored in a strand

![Figure 3. Overview of a DNA storage system.](image)
Interface and Addressing

- **Object Store**: \( \text{Put(key, value)} / \text{Get(key)} \).

- **Random access**: mapping a key to a pair of PCR primers.
  - **write**: primers are added to the strands
  - **read**: those same primers are used in PCR to amplify only the strands with the desired keys.

- **Separating the DNA strands into a collection of pools**:
  - primers reacts.
  - the chances of the sample contains all the desired data.
(a) The write process performs `put(key, value)`, generating a DNA library.

(b) The read process performs `get(key)` on a DNA library, returning the value.

**Figure 4.** Overview of a DNA storage system operation as a key-value store.
Encoding

• Base 4 encoding: 00, 01, 10, 11 => A, T, G, C.
  • Error prone: synthesis, PCR, sequencing (substitutions, insertions, and deletions of nucleotides)

• Base 3 + Huffman code + rotation code

(a) Translating binary data to DNA nucleotides via a Huffman code.

(b) A rotating encoding to nucleotides avoids homopolymers (repetitions of the same nucleotide), which are error-prone.
Data Format

Figure 6. An overview of the DNA data encoding format. After translating to nucleotides, the stream is divided into strands. Each strand contains a payload from the stream, together with addressing information to identify the strand and primer targets necessary for PCR and sequencing.
Adding Redundancy

**Goldman Encoding**

**XOR Encoding**

**Figure 7.** An encoding proposed by Goldman et al. [10]. The payloads of each strand are overlapping segments of the input stream, such that each block in the stream appears in four distinct strands.

**Figure 8.** Our proposed encoding incorporates redundancy by taking the exclusive-or of two payloads to form a third. Recovering any two of the three strands is sufficient to recover the third.