DSC-204A: Scalable Data Systems, Winter 2024

File System, Database, Cloud Storage

Lecturer: Hao Zhang *Scribe: Hena Ahmed, Jahnavi Patel, Abdullah Ashfaq*

Today's lecture marks the end of the second part of this class about cloud computing. Next week will begin the third of the four parts of the class, which will be about big data (specifically, parallelism and big data processing).

1 Recap: Collective Communication

1.1 Pros

As we learned in previous lectures, collective communication models typically use minimum-spanning tree (MST) or ring algorithms, depending on the size of messages being transmitted. The defining advantages of collective communication algorithms include:

1. **Well-structured communication primitives**, where each message can be processed with a common beginning and ending format.

2. **Easy performance analysis** and tracking due to the well-defined structure.

3. **Well-optimized** by hardware vendors (e.g., IBM, Nvidia) who have been researching and developing optimized collective communication for decades, and now even provide access to their technology through libraries such as MPI and NCI.

4. **Easy to program** because collective communication is an abstraction over peer-to-peer communication, which means that users only have to interact with a single API in order to make calls that will use collective communication.

1.2 Cons

Clusters require fault tolerance to retain stability in the event that a subset of its nodes fail. Additionally, clusters often have heterogeneous hardware requirements and setup to meet the computing needs. This leads us to the primary cons of collective communication:

1. **Lack of fault tolerance**, as each node depends on the other nodes in the collective to function.

2. **Requires homogeneity** in the computing environment so that each node can perform the same as one another.

Some alternatives to collective communication include peer-to-peer (P2P) and remote procedure calls (RPC), which have greater fault tolerance, which will be covered later in the class.

2 Storage

Storage is an ongoing area of research, as companies are trying to build better Platforms as a Service (PaaS) to meet cloud computing needs. The two categories of storage that we discuss today are: file systems/databases (specifically distributed systems), and cloud storage.

3 Filesystem

3.1 Files

A database is responsible for managing files, so it is important to be able to classify what a "file" and its defining components are. A **file** is an abstraction over bits of data, with a **file format** that contains metadata instruct applications on how to read the bits.

Below are the precise definitions for these features:

1. **File**: a persistent sequence of bytes that stores a logically coherent digital object for an application.
2. **File format**: an application-specific standard that dictates how to interpret and process a file's bytes.
3. **Metadata**: summary or organizing information about file content (i.e., the payload) stored with the file.

3.2 Storage of files

Files, which contain a combination of data and metadata, are stored and organized within the **file directory** of a computer for easier navigation. The director differs from the **file system**, which is component of the operating system that has a user-facing logical level and a physical level that communicates with underlying firmware.

Below are precise definitions for these storage concepts:

1. **Directory**: a cataloging structure with lists of references to files or recursive references to other directories.
2. **Filesystem**: the part of the OS that helps programs create, manage, and delete files on disk.

3.3 Differences and Unification

There are many different filesystems depending on OS. Some famous ones are ext2, ext3, NTFS. NTFS is the most adopted on windows. They differ on:

- How they layer files and directories and what metadata is stored to interpret the application
- How data integrity/reliability is assured, support for editing/resizing, compression/encryption, etc.

10 years ago when people were still debating which OS to adopt, the differences between filesystems were more but today these have become more unified due to the adoption of cloud computing. In cloud computing, we have multiple VMs each can have a different OS. We don't want to change filesystem each time we

switch VMs and want our storage devices to be mounted ("Mounted" is a system call in unix) on whatever VM we are using. So filesystems had to become more unified.

Question: What is a database? How is it different from just a bunch of files?
Answer: Database is more structured. We can search and retrieve a file in a fast way. If we put data in a bunch of files, it will be our own duty to search and retrieve. Database is essentially a collection of files but it is a manager and provides extra capabilities. These are:

1. **Maintenance:** Make sure your files are maintained and you won't lose them
2. **Performance:** Insert, delete, create, search are very fast
3. **Usability:** Easy to operate. High level APIs
4. **Security & Privacy:** If someone doesn't have access, it will prevent them from accessing data

3.4 Files vs Databases: Data Model

Database is an abstraction on top of the data files. Database manages at two levels: logical and physical.

- At **logical level**, database has a data model. Whenever we read something from database, we can interpret data in a certain format.
- At **physical level**, database manages how bytes are stored on disk.

All data systems (RDBMS, Dask, Spark, TensorFlow, etc) are application/platform software that use OS System Call API to handle data files.

Logical Level: How database interprets the data There are many different format but can be divided into two categories. These are commonly incorrectly used in industry:

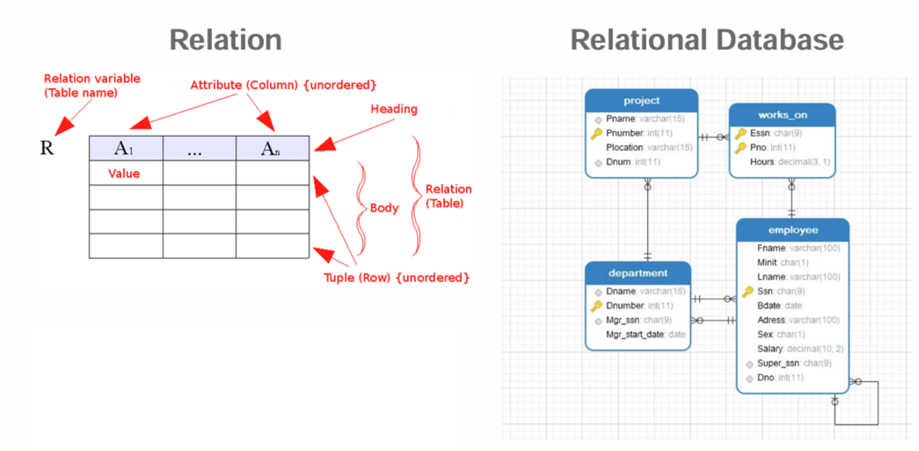
1. Structured data
2. Unstructured data

It is hard to distinguish between these. Different companies brand their products as capable of dealing with both structured and unstructured but if you check their product menu, their definition of these terms is different from other companies. This is my understanding but you should develop your own understanding of how to distinguish between them.

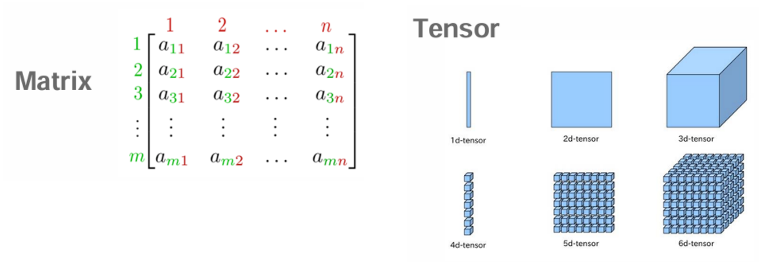
3.5 Data as File: Structured Data

Structured data is a form of data with regular substructure. Following are some examples:

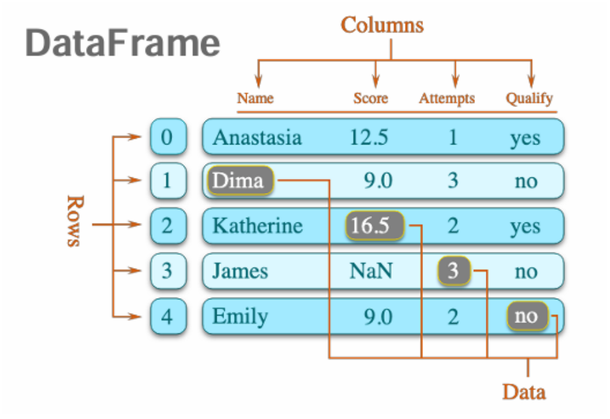
1. **Relation:** relation is structured. In relational database, we have a table which contains keys and attributes. Every data record is a row. Most RDBMSs and Spark serialize a relation as binary file, which is often compressed. It is the most developed format. Oracle is one of the most valuable companies in Silicon Valley which does business of this database.



2. **Matrix and Tensor:** Organized in dimensions. Became popular due to Machine Learning.



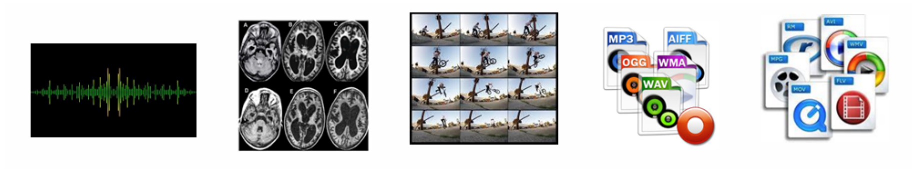
3. **DataFrame:** It is a little debatable if it is structured or unstructured. Today people say it's structured but 5 years ago, it was not the case. As compared to matrix and tensor, DataFrame is heterogeneous (each column can have different datatypes e.g. string, integer, file path). It organizes unstructured data in a structured way.
Snowflake and Databricks operate on DataFrame a lot in the form of Spark. DataFrame is typically serialized as restricted ASCII text file (TSV, CSV) and can layer relations too.



	Name	Score	Attempts	Qualify
0	Anastasia	12.5	1	yes
1	Dima	9.0	3	no
2	Katherine	16.5	2	yes
3	James	NaN	3	no
4	Emily	9.0	2	no

Below are some common formats for structured data (debatable as some are considered unstructured):

4. **Binary Formats:** Machine Perception data layer on tensors and time-series. There are binary formats to store images (jpeg, png), videos (mp4). Same formats support compression



5. **Text File:** Human-readable ASCII characters

6. **Docs/MultiModal File:** Myriad app-specific rich binary formats

Question: What is the difference between Relation, Matrix, and DataFrame?
Answer:

	Relation	**Matrix**	**DataFrame**
Ordering	Orderless on both axes	Has row/col number	Has row/col number
Schema Flexibility	Tuples have predefined schema	Cells are numbers	No pre-defined schema. All rows/cols can have names; col cells can be mixed types!
Transpose and Mathematical Operations	Can't transpose it. No mathematical operation associated with it.	Can transpose and matrix multiply.	Can transpose. Can multiply col/row elementwise but can't matrix multiply.

3.6 Relating Discussion with ChatGPT

Question 1: In what format are GPT-3 weights stored?
Answer: Each weight is a tensor but the connection can be stored in a dataframe or other higher level format.

Question 2: In what format are GPT-3 training data stores? Structured or Unstructured?
Answer: It's debatable but most people would say it's unstructured. Because GPT-3 has been trained on web documents (HTML) which has metadata and can't be stored in a structured way.

Rule of Thumb: Unstructured data are way more difficult to manage and deal with than structured data. How to process internet webpages for training models is a hot area and startups have been found to solve this. OpenAI crawled the entire internet (unstructured) to train GPT-3 which gives a very structured output. Data mining consumes structured and unstructured data to output some structured information.

4 Database

There are a lot of data structures in databases as it is a 50 years old industry. But we will be focusing on these important ones. Many companies are based on these.

4.1 Strawman

It is the simplest database with only get() and set() commands. It is append-only.

```bash
#!/bin/bash
db_set() {
    echo "$1, $2" >> database
    # $1 and $2 are key and value respectively provided by user to store in DB
}

db_get() {
    grep "^$1," database | sed -e "s/^$1,//" | tail -n 1
    # Explanation: It has 3 parts separated by pipe symbol i.e | which forwards result.
    # 1. Search the lines that start with a parameter $1
    # 2. Only output the value part
    # 3. Only output last line
}
```

Listing 1: Simplest Database (demo)

Properties

1. **Write (Append only):** We can't delete or modify existing value. Writing is efficient as we only search for location and write there. An application is database log which is append only.
2. **Read:** Inefficient as we have to scan entire database every time we search. Read time increases with file size linearly i.e. O(n).
3. **Some other Challenges:** This database has issues like concurrency (many people writing), disk space (as we are never deleting), handling errors (e.g. sudden shutdown)

Improvement: Indexing
To improve the issues, let's take example of library and see how books are organized. In the library, there are shelfs with tags. Each tag tells what the types of books on this shelf are. This is basically indexing.

Indexing is a natural mechanism in which we keep additional metadata on the side, which acts as a signpost and helps you to locate the data you want.
Pros:

1. Faster to find the data.
2. Update/remove/add the index is cheap as index doesn't scale linearly with number of records.

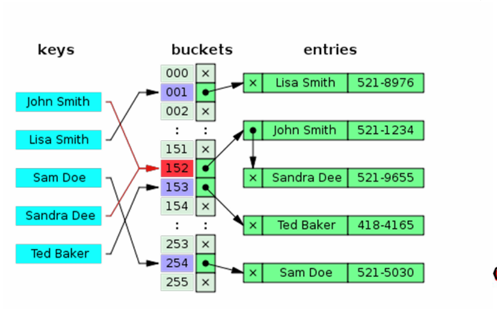
Cons:

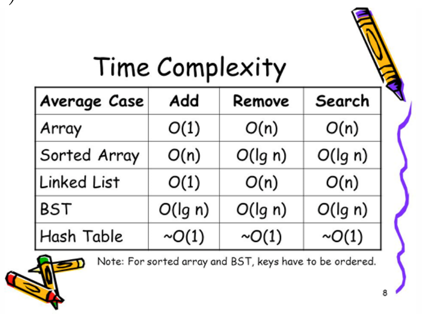
1. Writes become slower
2. Often index needs to be updated while writing.

Different index structure has different properties so we need to choose wisely. Choice is often dependent on domain knowledge and balancing of tradeoffs.

4.2 Hash Map/Table

First index that we will cover. It's safe to say that every database index is a HashMap. The most vanilla version of HashMap is a dictionary. We have a key and value and there is a 1-1 mapping between these. HashMap is very efficient. Search, insert, delete is $\sim O(1)$



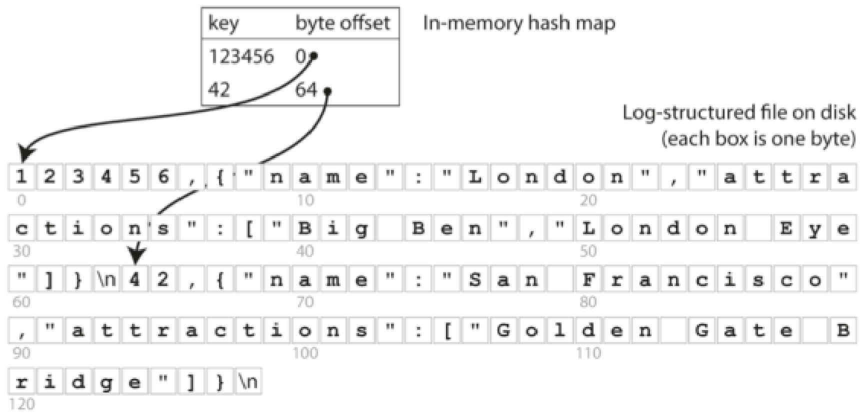

Time Complexity

Average Case	Add	Remove	Search
Array	O(1)	O(n)	O(n)
Sorted Array	O(n)	O(lg n)	O(lg n)
Linked List	O(1)	O(n)	O(n)
BST	O(lg n)	O(lg n)	O(lg n)
Hash Table	~O(1)	~O(1)	~O(1)

Note: For sorted array and BST, keys have to be ordered.

8

In python, dictionary is not database as it is in memory. In database, we put everything on disk. For database, we make a slight modification to HashMap that we store key and reference in memory but values are stored on disk

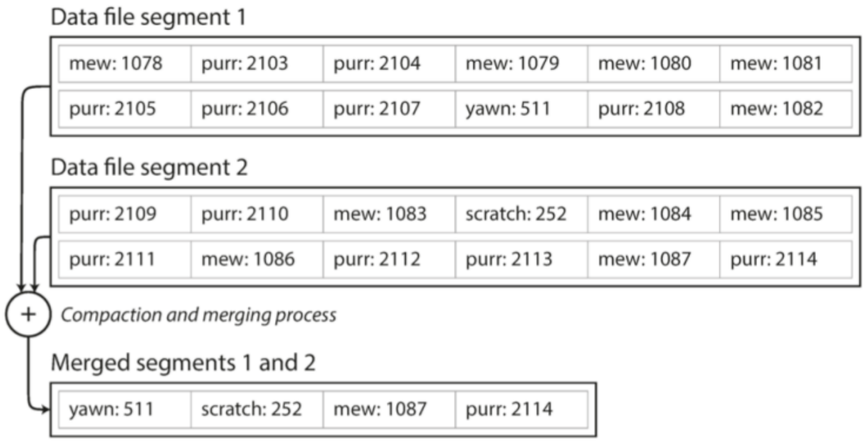

key byte offset
123456 0
42 64
In-memory hash map
Log-structured file on disk
(each box is one byte)
1 2 3 4 5 6 , { " n a m e " : " L o n d o n " , " a t t r a
c t i o n s " : [" B i g B e n " , " L o n d o n E y e
"] } \n 4 2 , { " n a m e " : " S a n F r a n c i s c o "
, " a t t r a c t i o n s " : [" G o l d e n G a t e B
r i d g e "] } \n
0
10
20
30
40
50
60
70
80
90
100
110
120

One application of this database is to monitor video play counts, where each click of the play button increments the value linked to the corresponding URL, acting as the key. However, scalability poses a challenge due to YouTube's extensive library of over 800 million videos. An approach to address this issue involves the implementation of SSTable.

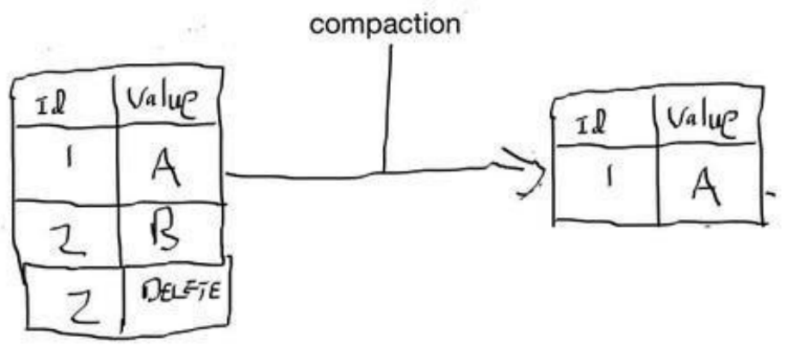
Improvement: Segment Compaction

Data is segmented into predetermined sizes. A background process, compaction, is executed to traverse the entire database, eliminating redundancy by discarding duplicate logs.

1. **Read/write compaction:** Read/write compaction operates on frozen segments already stored on disk, while background processes manage compaction within the database. The read/write compaction segment then replaces the old segment.



2. **Delete compaction:** Within the original segment, mark values for deletion after selecting the desired deletion target. The compaction process will remove the marked values, and the new segment will be saved with only the undeleted values. This approach ensures high efficiency.



Memory Crash Prevention:

When restarting the database, if the system encounters sluggishness, there's a risk of losing the index, making it hard to reconstruct from a sizable database segment.

Instead of loading the entire index, the system periodically transfers it from memory to disk. This ensures that even if the database shuts down, the index can still be retrieved from disk, comprising both the index and segment files.

However, if a power outage occurs during compaction, it could result in corrupted records. To address this issue, a checksum handling mechanism is implemented. Upon each database restart, records are read while their checksums are verified. If a checksum doesn't match its corresponding record, the system deletes the corrupted segment.

Pros:

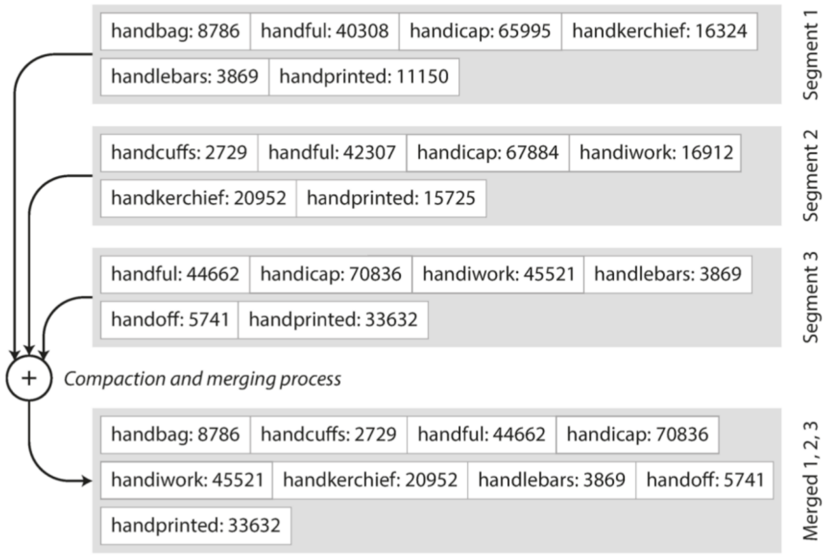
- Very fast writing speed.
- Simple concurrency and crash recovery, as there is no need to worry about partially written records.
- Avoids the problems of fragmented data files.

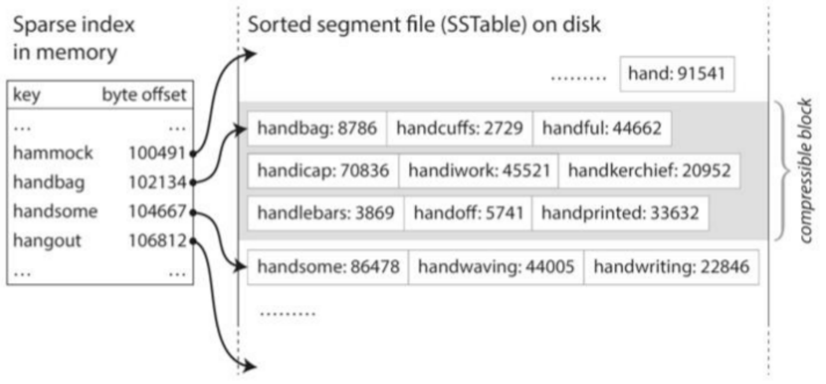
Cons:

- The hash table index must fit in memory.

4.3 SSTable

Minimizing the number of keys stored in memory is a crucial aim of SSTable. Achieving this involves sorting each segment's records based on their keys. By doing so, instead of storing every key in memory, we can store a sparse set of keys.





First, scan the memory to find the nearest left and right records; this guarantees finding the record when scanning left to right, reducing the need to scan the entire disk. Each segment file is relatively small, ranging from a few KB to MBs.

If keys and values had fixed sizes, binary search on the segment file could be used to avoid using an in-memory index. However, this approach would be inefficient since we cannot predict what will be written to the database. Sorting tables is advantageous as it allows for compression; keys are sorted alphabetically, meaning they share many prefixes, making compression easy as values with the same prefixes can be placed into the same segment.

Ensuring that each record in the database on disk is sorted can be achieved in two ways.

- One approach is to implement a background process that continuously sorts segments on disk until the entire database is sorted. However, this method is slow, particularly as the database grows larger.
- The second approach involves using a Memtable. Data manipulation occurs primarily in memory rather than on disk, including both values and keys. As users append new values, they are added to memory, allowing the database to grow until it reaches a predetermined cap, which represents the segment size.