VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

----------o0o----------

ASSIGNMENT REPORT

DATA ENGINEERING

MULTIMEDIA STORAGE AND PROCESSING WITH MONGODB

Instructor: Dr. Phan Trọng Nhân

Team: 2

Mai Chí Bảo - 2370691

Bùi Tuấn Anh - 2370688

Huỳnh Nhật Anh - 2370689

TP. HỒ CHÍ MINH, THÁNG 4 NĂM 2024

1. General theory

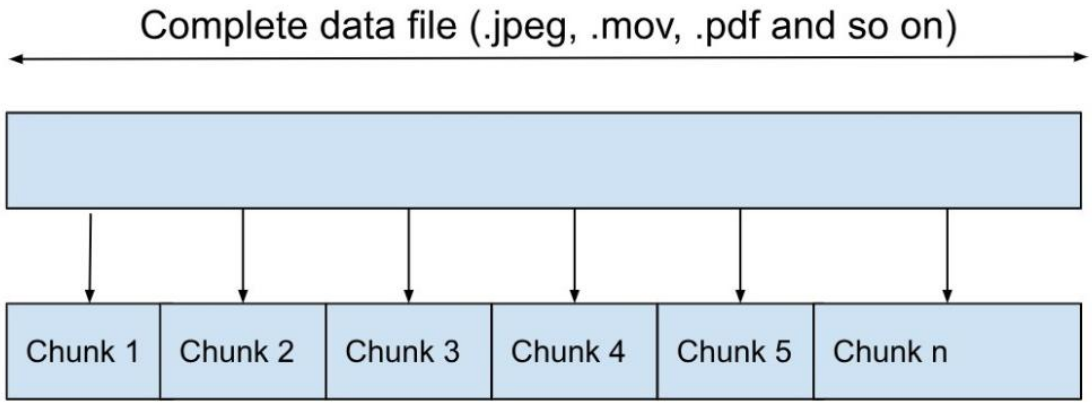
1. MongoDB

MongoDB is a leading NoSQL database management system renowned for its adaptability, scalability, and robust data storage and querying capabilities. Developed by MongoDB Inc., MongoDB revolutionizes data management with its document-oriented data model, allowing users to store data in flexible JSON-like documents rather than rigid tables with fixed schemas. One of MongoDB's key features is its ability to scale effortlessly, both vertically and horizontally. Vertical scaling involves increasing the capacity of a single server, while horizontal scaling involves adding more servers to distribute the load. This scalability makes MongoDB suitable for handling large volumes of data and high traffic applications. Another strength of MongoDB lies in its support for unstructured and semi-structured data. Unlike traditional relational databases that require predefined schemas, MongoDB allows for dynamic and schema-less data structures. This flexibility is particularly beneficial in scenarios where data schemas evolve over time or where dealing with diverse data types is necessary. Moreover, MongoDB excels in performance, providing fast and efficient querying capabilities. Its support for indexes, including compound indexes and geospatial indexes, enhances query performance even further.

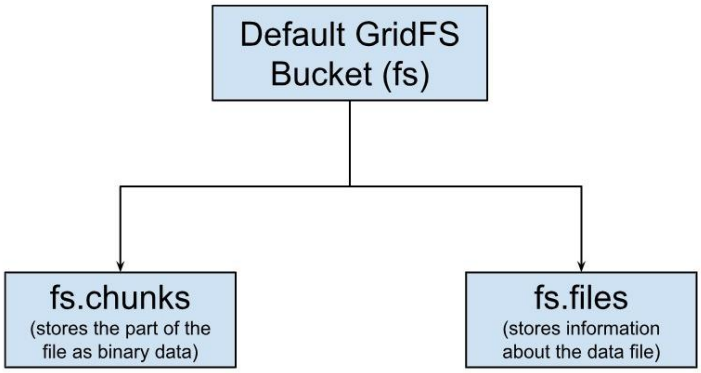
In summary, MongoDB stands out as a versatile and powerful database solution that meets the demands of modern applications. Its flexibility, scalability, performance, and ease of use make it an ideal choice for organizations seeking to manage and analyze vast amounts of data efficiently and effectively. We therefore want to discover the performance of MongoDB when dealing with storing multimedia data such as videos as discussed and analyzed below.

2. GridFs

Instead of storing a file in a single document, GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. By default, GridFS uses a default chunk size of 255 kB; that is, GridFS divides a file into chunks of 255 kB with the exception of the last chunk. The last chunk is only as large as necessary. Similarly, files that are no larger than the chunk size only have a final chunk, using only as much space as needed plus some additional metadata.



GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata. The section GridFS Collections describes each collection in detail.



The example of a document stored in the fs.chunks collection. The _id field here is the unique identifier for the particular chunk of the file. The files_id represents the actual file, the name of which is associated in the fs.files collection. The number ‘n’ represents the position of the chunk in the entire file,

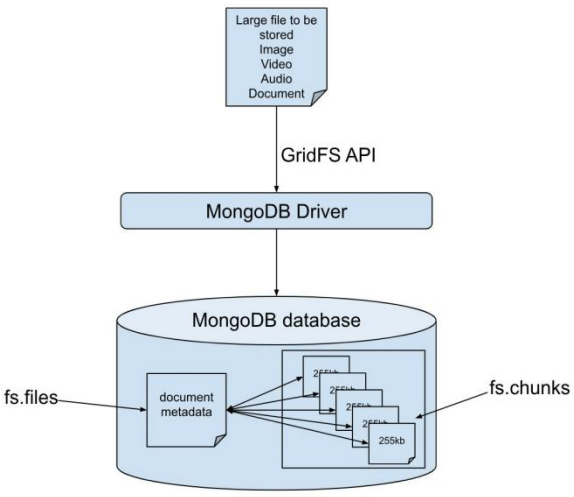
and is in sequence from 0 to n-1. ‘data’ represents the actual chunk of data stored in the particular document.

```
{
    "_id": ObjectId("554a76f34f54bfzc8a3ex04d"),
    "files_id": ObjectId("223a75g19l94bxel8p2ef32y"),
    "n":NumberLong(0),
    "data": BinData(0, "+MCI+Pxf5u93g...........")

}
```

The example of metadata: The _id field is the unique identifier for the file. The length field represents the total length of the file. As MongoDB offers a flexible schema, we can also add additional fields

```
{
  "_id": ObjectId("223a75g19l94bxel8p2ef32y"),
  "length": NumberLong("998321"),
  "content-type": "video/quicktime",
  "uploadDate": ISODate("2023-09-20T23:00:01Z"),
  "chunkSize": NumberLong("34221"),
  "filename": "star_wars.MOV",
  "author": "misc"
}
```

When query GridFS for a file, the driver will reassemble the chunks as needed. You can perform range queries on files stored through GridFS. We can also access information from arbitrary sections of files, such as to "skip" to the middle of a video or audio file.



While inserting data, GridFS checks for the MD5 checksum to ensure that the chunks belong to the same file.

It also creates default indexes to ensure the efficiency and speed of the assembly and retrieval of the binary documents.

3. Ways to save Multimedia with MongoDB

a) Direct Embedding in MongoDB Documents

Advantages:

- Simplifies data retrieval and management by encapsulating data within the document.
- Suitable for smaller multimedia files or applications with lower storage demands.
- Eliminates the need for multiple storage systems management.
- Supports advanced queries and replication.

Drawbacks:

- Increases the size of documents, potentially affecting query performance and storage costs for large files.
- Uses BSON format with a 16MB limit per document; larger files require GridFS which chunks files into segments under 215KiB each.
- Not optimized for real-time streaming or processing of large multimedia files.

b) Storing File Paths in MongoDB and Multimedia Externally

Advantages:

- Reduces storage overhead on MongoDB, improving query performance for large files.

- Facilitates integration with external storage solutions or CDNs, enhancing scalability and flexibility.
- Suitable for handling larger multimedia files or high-performance requirements.

Drawbacks:

- Adds complexity in managing data consistency between MongoDB and external storage.
- Requires additional mechanisms for data synchronization to ensure integrity.
- May involve higher costs and effort in maintaining connections with third-party services.
- General Considerations

c）General

Performance and Scalability:

- Optimal database schema design and efficient indexing strategies are crucial for maintaining performance.
- Alternative storage solutions should be considered based on specific data types, sizes, and use cases.

Resource Management:

- Careful planning and resource allocation are necessary to effectively handle the storage and management of multimedia data.

4. Other approaches to save Multimedia data

a) Amazon S3

AWS S3 is a highly scalable, secure, durable, and cost-effective object storage service provided by Amazon Web Services (AWS). It allows users to store and

retrieve any amount of data from anywhere on the web, offering a simple web service interface to store and retrieve data.

S3 is designed to provide 99.999999999% (11 nines) durability of objects over a given year and 99.99% availability of the service. Objects stored in S3 are organized into buckets, which are similar to folders or directories. Each bucket must have a unique name across the entire AWS S3 namespace.

S3 supports a virtually unlimited number of objects and can store objects ranging in size from a few bytes to terabytes.

A bucket in Amazon S3 serves as a container for storing objects. Each bucket can hold an unlimited number of objects and is uniquely identified by a name.

Objects are the fundamental entities stored within Amazon S3. Each object consists of data and metadata, with metadata comprising name-value pairs describing the object's properties. Objects are uniquely identified within a bucket by a key (name) and, optionally, a version ID if S3 Versioning is enabled. Key features of objects include: Unique Identification, Metadata, Addressing

S3 Versioning is a feature that enables users to keep multiple variants of an object within the same bucket. With versioning enabled, every modification to an object results in a new version being created, allowing users to preserve, retrieve, and restore previous versions as needed. Key features of S3 Versioning include: Data Preservation, Unique Version ID

b) Amazon S3 and MongoDB in saving Multimedia

i. Performance

The NoSQL nature of MongoDB makes data operations quick and easy. Without compromising the data integrity, data can be quickly stored, updated and retrieved.

AWS S3 provides actions such as multi-Part payload which gives low latency of 100–200 milliseconds which automatically grows to high request rates.

Resources GridFS, a specification for storing and retrieving large files such as images, video files, and audio files with MongoDB, can impact performance based on the memory and processor capabilities of the hosting server. For smaller files accessed infrequently, GridFS might exhibit slightly superior performance to AWS S3 due to its streamlined operation within the database itself. However, for larger files or higher frequency access, S3's dedicated infrastructure might provide better performance due to its optimized data retrieval paths and global distribution networks.

The location of the storage platform relative to the application hosting environment plays a critical role in determining the actual performance experienced by end-users:

Amazon Hosted Applications: For applications running on the Amazon ecosystem, S3 offers lower latency due to optimized paths within the Amazon network, making it a superior choice for such setups.

On-Premises Applications: When the application is hosted on-premises, the choice between MongoDB and S3 becomes more nuanced. A MongoDB instance on the same network as the application would likely outperform S3 due to the absence of significant network latency.

Internet-Transferred Data: For applications where data must be transferred over the internet to a managed MongoDB instance or to S3, the network quality and distance can significantly impact performance. In such cases, S3's global infrastructure potentially offers more consistent and faster data access, particularly for applications already hosted within the Amazon ecosystem.

Processor and Memory Costs: MongoDB's architecture demands more in terms of memory and processing power compared to AWS S3. This is particularly relevant when dealing with the storage of large files at a high frequency. AWS S3, designed as an object storage service, is optimized for such tasks and thus incurs lower costs related to processing and memory usage. Therefore, for high-volume storage needs, S3 emerges as a more cost-effective choice.

Disk Space: The cost of disk space is another critical factor. If the primary concern is the storage of large files without a pressing need for speed, the decision boils down to the comparative costs of disk space. This requires a personalized calculation, considering the specific costs of disk space for your MongoDB setup versus the pricing tiers of AWS S3. For those who can manage with slower access speeds, the choice may hinge on which option offers the more economical storage per gigabyte.

ii. Data type

BSON: which stands for Binary JSON (JavaScript Object Notation), is a binary-encoded serialization of JSON-like documents. It is designed to be efficient in space and speed when encoding and decoding data in MongoDB, a popular NoSQL database. BSON extends the JSON model to provide additional data types and to be efficient for encoding and decoding within different languages. Efficient Storage and Retrieval: BSON is designed to be more efficient than JSON in terms of storage space and scan-speed. It allows MongoDB to store types not represented in JSON, such as Date and binary data.

Object Files: In the context of Amazon S3, an object file refers to any piece of data stored in S3. An S3 object can be any file type - from simple text files and images to complex applications - stored in a flat, scalable storage structure. Unlike BSON, which is a specific data format, S3 objects are essentially binary files without any imposed structure, identified and accessed through keys (file

names) within buckets (storage containers). Binary Data Format: S3 objects store data in a raw binary format. This means that S3 can store files in their native formats, be it images, videos, application data, or any other file type, without needing a specific data structure or format

iii. Data storage

MongoDB Storage Characteristics:

Data Model: MongoDB is a document-oriented database that stores data in JSON-like (BSON) documents, allowing for a flexible and dynamic schema. This flexibility is advantageous for storing text and complex hierarchical data, which can change over time.

Storage Organization: Data in MongoDB is organized into collections and databases. Collections can be thought of as equivalent to tables in a relational database, but without a fixed schema. This organization suits applications that require rapid iteration and where data structures can evolve.

Suitability for Data Types:

- Text: MongoDB's document model is inherently suited for storing and querying text documents, including JSON, XML, or BSON data, making it a good choice for content management systems, blogging platforms, or any application that handles a lot of variable text data.

- Images and Videos: While MongoDB can store binary data (such as images and videos) using GridFS (when files exceed the BSON-document size limit of 16MB), it's generally not as efficient or cost-effective as using a dedicated object storage service like S3 for large or numerous multimedia files.

Amazon S3 Storage Characteristics:

Data Model: S3 is an object storage service that treats data as objects. Each object is stored in a bucket and is identified by a unique key. Objects can

contain data in any format (binary), making S3 extremely versatile for storing a wide variety of file types.

Storage Organization: S3 organizes data into buckets, which are similar to top-level directories. Within each bucket, objects can be further organized with keys that simulate a hierarchical structure (e.g., folder1/subfolder1/file1). Additionally, S3 data is stored across multiple facilities within a region, ensuring high durability and availability.

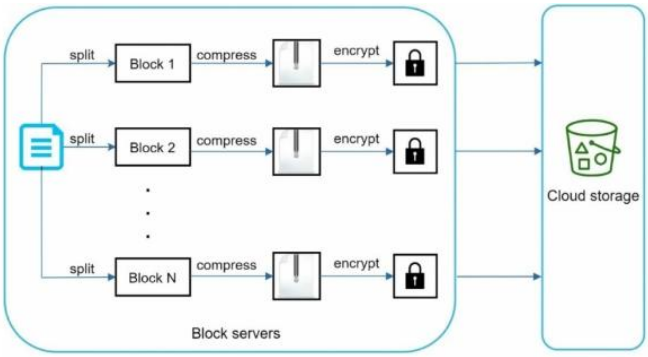
Suitability for Data Types:

- Text: S3 can efficiently store large quantities of text data, such as logs or raw text files. However, for applications that require complex queries or frequent updates to text data, the flexibility of MongoDB might be more appropriate.

- Images and Videos: S3 is particularly well-suited for storing and serving large files such as images and videos. Its high durability, availability, and built-in features like versioning and lifecycle management make it an ideal choice for media hosting, content delivery networks, and data archiving.

iv. Data Preprocessing

In s3 A file is split into smaller blocks. Each block is compressed using compression algorithms. To ensure security, each block is encrypted before it is sent to cloud storage. Blocks are uploaded to the cloud storage



Instead of storing a file in a single document, GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. By default, GridFS uses a default chunk size of 255 kB; that is, GridFS divides a file into chunks

of 255 kB with the exception of the last chunk. The last chunk is only as largeas necessary.

v. Scaling

As the data grows, MongoDB quickly and equally distributed the data across a cluster of computers. This process is called "Sharding". Growing amounts of data are easily handled with MongoDB's scalability.

There is no set limit on the volume of data or the number of objects that can be stored in an S3 bucket. S3 offers infinite scalability. A bucket created in s3 has a 5T storage limit

Horizontal Scaling involving the expansion of storage space and processing nodes.

MongoDB: Horizontal scaling with MongoDB necessitates increased processor and memory resources as additional nodes are added to the cluster. This scaling approach can lead to escalating costs, particularly as the infrastructure grows. Consequently, the cost-effectiveness of MongoDB diminishes with the expansion of nodes.

Amazon S3: Scalability with Amazon S3 is generally more cost-effective due to its native support for horizontal scaling and auto-scaling capabilities. S3's infrastructure allows for seamless expansion of storage space and processing capacity as demand grows.

Vertical scaling involving increasing the capacity of individual nodes.

MongoDB: As the pressure on the storage system increases, the cost of vertical scaling with MongoDB may become more significant. MongoDB requires additional processor and memory resources to handle higher loads efficiently. This approach can lead to higher costs as the infrastructure needs to be continuously upgraded to meet growing demands.

Amazon S3: Vertical scaling considerations with Amazon S3 are less pronounced compared to MongoDB. S3's architecture abstracts away the underlying infrastructure, allowing for seamless expansion of storage capacity without the need for manual adjustments to individual nodes. However, as with any cloud service, costs may still increase linearly with storage usage, but without the need for upfront investment in hardware upgrades

2. Demo

To demonstrate how to use MongoDB for multimedia we must use this database management system as this is our proposal to store multimedia files.

Setting up AWS S3 for storage

AWS IAM for secret key and access via API.

To interact with blob storage and integrate more efficiently with Mongodb you will need a programming interface that can help you to program. AWS has provided us with every means to interact with them.

To generate an AWS secret key, you can follow these steps:

- Log into your AWS Management Console.
- Navigate to your account details by clicking on your profile name at the top right corner.
- Select"Security Credentials" from the dropdown menu.
- In the Security Credentials page, scroll down to the"Access keys (access key ID and secret access key)" section.
- Click on"Create New Access Key".
- A popup will appear with your new access key ID and secret access key. Click on"Show Access Key" to view the secret access key.
- Download the key file or copy the access key ID and secret access key and store them securely. You won't be able to retrieve the secret access key again after this dialog box is closed.

Remember, it's important to keep your secret access key confidential to protect your AWS account.

Creating a bucket in AWS S3 involves a few simple steps:

- Sign in to AWS Management Console: Log into your AWS account.

- Open the Amazon S3 Console: You can find it under the "Services" menu or search for S3 in the search bar.
- Create a New Bucket: Click on the "Create bucket" button.
- Set Bucket Name and Region:
- Bucket Name: Choose a unique name for your bucket.
- Region: Select the AWS Region where you want your bucket to reside.
- Configure Options(Optional): Set up options like versioning, logging, tags, etc., if needed.
- Set Permissions: Configure the permissions for your bucket. You can set up ACLs (Access Control Lists) or use bucket policies to manage access.7. Review and Create: Review your settings and click "Create bucket" to finalize the bucket creation.

Remember, bucket names must be unique across all existing bucket names in Amazon S3. Once created, you cannot change the name of the bucket or its Region. For detailed guidance, you can refer to the official AWS documentation.

Tech stacks and libraries

In this demo, we used

- express.js: To export the interface to MongoDB as API.
- multer: Multer is a Node.js middleware that handles multipart/form-data, primarily used for uploading files with maximum efficiency.
- multer-girdfs-storage: Multer-GridFS-Storage is a storage engine for Multer that allows you to store uploaded files in MongoDB's GridFS.
- Aws-sdk: The AWS SDK is a collection of libraries and tools that enable developers to build, deploy, and manage applications integrated with Amazon Web Services across a variety of programming languages.

- multer-s3: Multer-S3 is a Node.js middleware for handling multipart/form-data, specifically designed for uploading files to Amazon S3.
- mongoose: Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js, providing schema validation and data relationship management.
- dotenv: Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a web browser.

To demonstrate with s3. we will declare a simple mongoose model that can capture basic information of the image that is stored on aws s3. To store these informations we have to store the link of the image on aws s3 and file's name for convenient querying.

```
const VideoSchema = new Schema({
    title: {
        type: String,
        required: true
    },
    storageName: {
        type: String,
        required: true
    },
    fileName: {
        type: String,
        required: true
    },
    // change type to string if your uploading just one image
    url: {
        type: String,
        required: true
    },
    version: {
        type: String,
        required: false
    },
    createdAt: {
        type: Date,
        default: Date.now
    },
});
const Video = mongoose.model("Video", VideoSchema);
```

Then in our code we will have to write the code that integrate with mongodb and aws s3 storage at the same time. With the help of multer S3 library that can help us to process the files or make us easier in converting image to byte and upload to S3 at the same time.

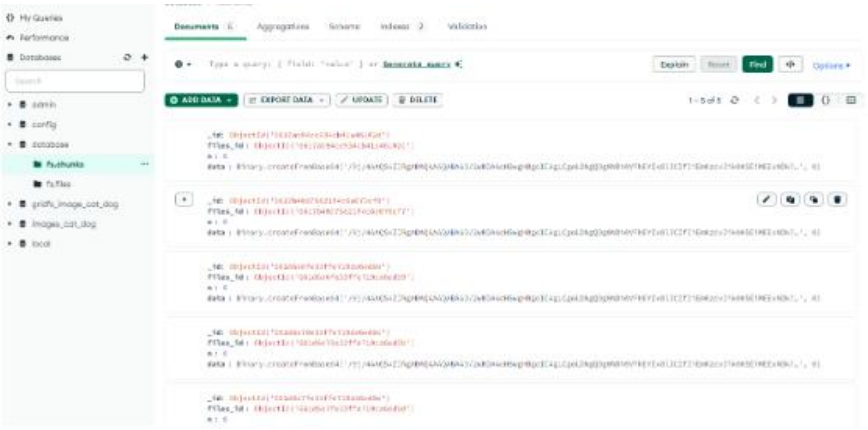
```
const upload = multer({
    storage: multerS3({
        s3: s3,
        bucket: 'dog-cat-bucket-mongo',
        metadata: (req, file, cb) => {
            console.log(file)
            cb(null, { fieldName: file.fieldname });
        },
        key: (req, file, cb) => {
            cb(null, Date.now().toString())
        }
    })
})
```

Finally, when we combine everything together. We will have the data as in this picture.

To demonstrate how to store images with gridfs of mongodb, we have used multer-gridfs-storage that handles the file data and save to mongodb with ease.

```
// Create a storage object with a given configuration
const storage = new GridFsStorage({ url: URL + DATABASE_NAME } );

// Set multer storage engine to the newly created object
const upload = multer({ storage });

const app = express();
const client = new MongoClient(URL, { useNewUrlParser: true, useUnifiedTopology: true });

// Upload your files as usual
app.post('/profile', upload.single('avatar'), (req, res, next) => {
        // TODO: Store fileId as support query as
    res.send("image uploaded successfully.");

    /*....*/
});
```

And the result will look like this.


My Queries
Performance
Databases
admin
config
database
fs.chunks
fs.files
gridfs_image_cat_dog
images_cat_dog
local
Documents 6
Aggregations
Schema
Indexes 2
Validation
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
UPDATE
DELETE
1 - 5 of 5

3. Conclusion

In conclusion, MongoDB presents a robust solution for storing multimedia data, offering flexibility and scalability to meet the diverse needs of modern applications. By leveraging MongoDB's document-oriented data model, multimedia data can be stored directly within documents or managed externally with file paths, each method catering to specific requirements and trade-offs.

GridFS, MongoDB's file storage system, addresses the limitations of BSON document size by chunking large multimedia files into smaller segments, ensuring efficient storage and retrieval. This mechanism allows MongoDB to effectively handle multimedia files of varying sizes, making it a suitable choice for applications with diverse storage demands.

Comparing MongoDB's multimedia storage capabilities with Amazon S3, both offer unique advantages. MongoDB excels in providing seamless integration with application data, simplifying management and reducing overhead for smaller multimedia files. However, for larger files or scenarios requiring extensive metadata management, Amazon S3's scalable object storage service may offer better performance and cost-effectiveness.