Chap 1: Introduction to big data management and processing

A diagram of data analysis

AI-generated content may be incorrect.

Các stack trong lĩnh vực Big Data

- Một hệ thống big data cần đảm bảo các tiêu chí sau:

* Scalability: có khả năng nhân rộng hệ thống khi lượng data tăng
* Accessibility: có khả năng ghi/đọc data thuận tiện
* Transparency: user có thể truy cập data dễ dàng như khi được lưu local và không nên biết được vị trí physical của data
* Availability: khả năng duy trì mặc dù có những lỗi nhỏ

- Curse of dimensionality: khi có càng nhiều feature hơn, thì cần nhiều lượng data để tăng độ chính xác. Nếu giữ nguyên lượng data mặc dù tăng feature thì độ chính xác của classifier sẽ giảm

A graph with a blue line

AI-generated content may be incorrect.

Chap 2: Hadoop Ecosystem

Trong Hadoop, sẽ có các hệ thống chính sau:

* Lưu trữ dữ liệu: hệ thống tệp tin phân tán Hadoop (HDFS)
* Xử lý dữ liệu: MapReduce Framework
* Hadoop Common: chứa các tiện ích của Hadoop
* Hadoop YARN: quản lý tài nguyên và lập lịch

Trong Hadoop, dữ liệu được lưu dưới dạng distributed. Trong đó mỗi máy tính tham gia cả 2 vai trò là lưu trữ và tính toán (được gọi là 1 node)

- Để có thể xử lý lỗi mà không mất dữ liệu, Hadoop dùng kỹ thuật “dư thừa”:

* Khi một node bị lỗi => dữ liệu node đó được nhân bản trên các node còn lại (trong hadoop, các tệp tin được phân mảnh để lưu trên các nodes)
* Khi xử lý 1 tác vụ a, tác vụ a này được phân mảnh thành các tác vụ độc lập (để thực thi song song trên từng mảnh ở các nodes khác nhau). Tác vụ bị lỗi sẽ được tái lập lịch trên một nodes khác.

1. HDFS:

- Nền tảng lưu trữ dữ liệu lớn của Hadoop

- Có thực hiện phân quyền cho người dùng

- Write one, read many: khi dữ liệu đã được viết vào file => chỉ có thể đọc chứ không thể sửa hay ghi thêm nữa => muốn thay đổi phải ghi đè or viết thêm. Lý do áp dụng:

* Big data thường là dữ liệu không cần thay đổi
* Phù hợp cho việc đồng bộ dữ liệu giữa các nodes (do mỗi file thường được nhân bản nhiều trong hdfs)
* Giảm phức tạp khi đồng bộ

- HDFS thiết kế các nodes theo kiến trúc master/slave:

* Master (name nodes): lưu ánh xạ dữ liệu tới vị trí nodes chứa dữ liệu đó (meta data) và lưu không gian tên
* Slave (datanodes): lưu trữ trực tiếp dữ liệu

A diagram of a data structure

AI-generated content may be incorrect.

Cấu trúc master/slave

- Đặc điểm của HDFS như sau:

* Dữ liệu được chia thành các chunk (64 MB) và lưu trên các nodes
* Chỉ có được ghi thêm vào file (append)
* Mỗi chunk được nhân bản 3 lần (phòng th lỗi)
* Để xử lý lỗi:
  + Data node: Nhân bản 3 lần mỗi chunk
  + Name node: S/d secondary name node

2. Mapreduce:

- Là phương thức xử lý dữ liệu trong Hadoop, được đề xuất bởi Google

- Trong mapreduce, mỗi tác vụ tổng (job) được phân tán thành nhiều tác vụ nhỏ (task) để thực thi song song trên các nodes. Các tasks được thực hiện song song và độc lập => khả mở

- Ta viết code theo phương thức mapreduce, sau đó đoạn code này sẽ được gửi cho từng nodes có chứa dữ liệu tương ứng. Đoạn code map reduce được viết gồm 2 hàm Map và Reduce (input và output đều sẽ là các cặp key – value)

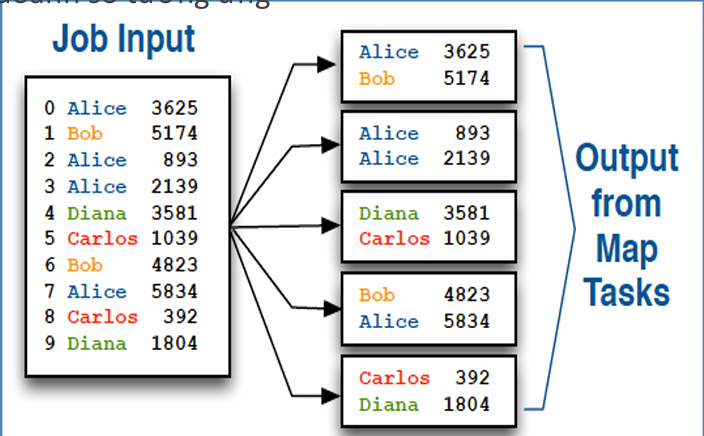
A diagram of a job output

AI-generated content may be incorrect.

V/d đầu vào/đầu ra của một đoạn code Map Reduce

Các bước trong Map Reduce:

- Mapping: Mỗi mapping sẽ xử lý một phần dữ liệu (1 chunk). Nhận vào 1 chunk và đưa ra 0 or nhiều cặp key – value trung gian



Ví dụ về mapping => đưa ra tên nhân viên và doanh số tương ứng

- Shuffle và sort: output của các map tasks sẽ được sắp xếp lại dựa vào key để đưa vào reduce

A computer screen with multiple colored lines

AI-generated content may be incorrect.

Ví dụ về shuffle and sort

- Reduce: output của shuffle và sort được đưa vào reduce. Trong reduce, mỗi key sẽ chỉ được xử lý bởi một reducer duy nhất => reducer này sẽ xét từng value tương ứng của key một.

A screenshot of a computer

AI-generated content may be incorrect.

Ví dụ về Reduce

Ví dụ về phương thức map reduce trong bài toán đếm từ:

A diagram of a flowchart

AI-generated content may be incorrect.

Trong môi trường distributed, shuffle thực tế là phương thức giúp trao đổi data giữa các nodes => ghép lại theo key

A diagram of a network

AI-generated content may be incorrect.

Thông thường, client program sẽ gửi dữ liệu input + job cần thực hiện (mã map + reduce) cho hadoop để thực hiện. Job này sẽ được gửi đến job tracker (năm ở master node (name node) trong hdfs). Job tracker này sẽ chia job thành các task của task tracker nằm ở các slave nodes và quản lý trình tự thực hiện các task (có thể lặp lịch lại nếu một task bị lỗi). Task tracker: nằm trên các slave nodes, giúp thực thi hàm map và reduce đối với dữ liệu lưu trên nodes

A diagram of a task tracker

AI-generated content may be incorrect.

Chi tiết của quá trình map reduce trên môi trường phân tán sử dụng job và task tracker

3. Apache Pig:

- Giao diện giao job cho hadoop ở mức cao (thân thiện với người dùng) => tự động biến đổi các pig script thành jobs để thực hiện map reduce trên HDFS

A blue background with black text

AI-generated content may be incorrect.

Một pig script giúp tính order cost của mỗi người dùng

4. Apache Hive:

- Ngoài apache pig ta cũng có apache hive (vai trò hoạt động giống hết apache pig). Tuy nhiên hive script bản chất là lệnh sql => chuyển từ lệnh sql thành mapreduce jobs trên hadoop

5. Apache Hbase:

- Là một hệ quản trị csdl của hadoop

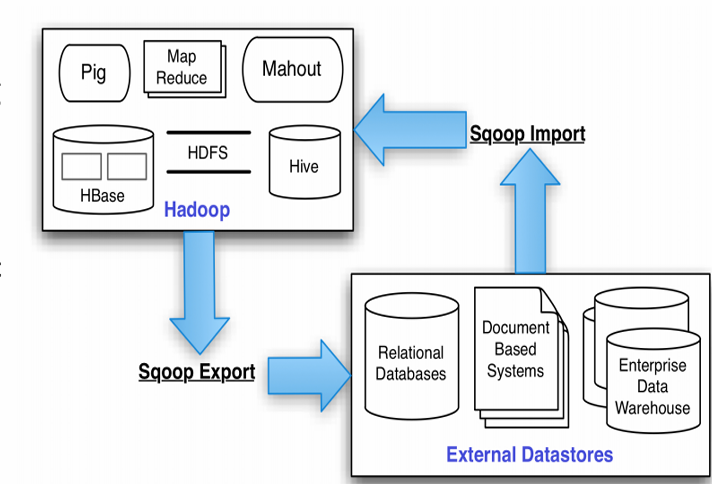
- Dữ liệu được lưu trữ dưới dạng bảng (gồm rất nhiều cột và dòng ) (do là big data)

=> Tuy nhiên, không hỗ trợ ngôn ngữ truy vấn sql. Nếu muốn truy vấn trực tiếp phải gọi API hoặc dùng truy vấn sql thông qua apache hive.

6. Apache Sqoop

- Là một tool giúp trao đổi dữ liệu giữa HDFS và csdl quan hệ (import /export vào HDFS)

- HDFS trao đổi với các csdl khác dựa vào các mapreduce job



Trao đổi dữ liệu = apache sqoop

7. Apache Kafka

- Nền tảng xử lý dữ liệu dạng luồng (distributed streaming platform), giúp truyền tải dữ liệu theo thời gian thực

A diagram of a company

AI-generated content may be incorrect.

Trong kafa, csdl được lưu ở broker. Producer sẽ truyền dữ liệu vào broker. Khi nào consumer cần truy cập vào dữ liệu thì sẽ truy cập vào broker

8. Apache Oozie:

- Là hệ thống lập lịch cho job -> quản lý các job trên Hadoop.

- Luồng wokflow lập lịch của oozie được biểu hiện bởi đồ thị vòng có hướng

- Một số cộng việc được hỗ trợ lập lịch trong oozie:

* Pig và Hive Script (để truy cập vào HDFS)
* Các map reduce jobs (thực hiện tính toán, import/export trong database)
* Đoạn code java hoặc shell
* Chạy ctr qua SSH
* Gửi/nhận email

9. Apache Zookeeper:

- Giúp điều phối, quản lý các máy chủ trong hệ thống phân tán:

* Quản lý các nodes
* Quản lý trạng thái nodes
* Quản lý thông tin cấu hình
* Bầu cử leader

A diagram of a server

AI-generated content may be incorrect.

Một hệ thống được quản lý bởi apache zookeeper

10. PAXOS Algo:

- Thuật toán đồng thuận giúp nhận được thông tin giá trị ngay cả khi có một số node bị lỗi.

A screenshot of a computer

AI-generated content may be incorrect.

Cách hoạt động của PAXOS

A screenshot of a computer

AI-generated content may be incorrect.

Ví dụ về PAXOS

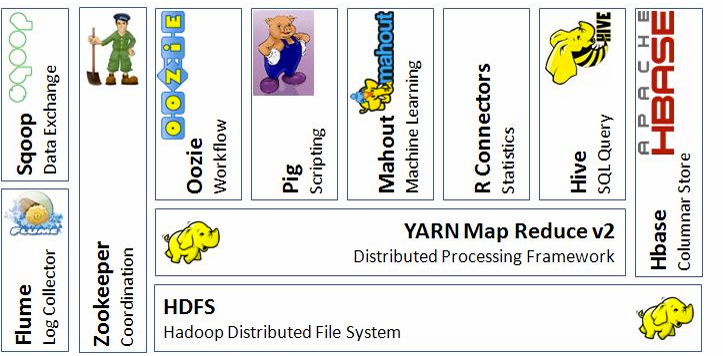
11. YARN:

- Hệ thống cấp phát tài nguyên phù hợp (bộ nhớ và CPU) của nodes cho các ứng dụng

A diagram of a resource manager

AI-generated content may be incorrect.

Phân chia tài nguyên bằng YARN



Chapter 3: Hadoop HDFS File System

1. Tổng quan về thiết kế hệ thống distributed:

Mục tiêu của một hệ thống distributed:

- Network Transparency: Mọi user truy cập file dễ dàng như ở local. User truy cập file mà không cần biết physical location của file.

- Transparency trong hệ thống

* Location Transparency: tên file không phản ánh vị trí physical của file
* Location Independence: file có thể tự do di chuyển giữa các physcal location (không cần đổi lại tên file).

Có thể thấy Localtion Independence => Location Transparency nhưng điều ngược lại không đúng.

- Availability: files được truy cập dễ dàng từ nhiều vị trí. Số lượng users, system failures không ảnh hưởng tới khả năng availability của file.

Các kiến trúc của Big Data:

- Client – Server: Một hoặc một số máy sẽ đóng vai trò là server (lưu trữ data, file). Request (operations cho data, query data) sẽ thực hiện từ client gửi đến server.

- Symmetric: Hệ thống đối xứng, mỗi máy vừa đóng vai trò là client vừa là server (peer to peer)

2. Một số vấn đề trong thiết kế hệ thống:  
2.1. Naming and name resolution:

- Name space: tập hợp các name để tham chiếu đến các object trong hệ thống

- Name resolution: quá trình ánh xa một name thành đối tượng thật

- Thông thường, sẽ có 3 cách để quản lý tên trong distributed sys:

* Gắn tên máy chủ vào đường dẫn đến file: VD: server1:/data/file1.txt
* Mount dữ liệu từ các máy chủ vào cây thư mục cục bộ

A black screen with white text

AI-generated content may be incorrect.

* Provide a single glocal directory: toàn hệ thống có một hệ thống file duy nhất => người dùng khi truy cập sẽ không cần biết file nằm ở host nào và khong biết physical location của file (cách này HDFS trong Hadoop dùng)

2.2. Semantics of file sharing:  
- Cần cơ chế xử lý concurrent read/write hợp lý

- Một số cơ chế khác nhau để sử lý concurrent read/write:

* UNIX: nếu một process write vào file => ngay lập tức các tiến trình khác đang mở cùng file sẽ thấy thay đổi đó
* Session semantics: nếu 1 process write vào file, các tiến trình khác không thấy thay đổi trên file. Chỉ khi file đó đóng thì hệ thống mới lưu thay đổi
* Immutable Shared Files semantics: file không thể thay đổi sau khi ghi. Muốn thay đổi thì phải tạo file mới
* Transaction semantics: Các nhóm read/write phải có tính all-or-nothing: hoặc thành công hoặc không có gì thay đổi. vd: p1: W1 w2 và p2: r1,r2 thì p2 phải đọc giá trị trước w1 w2 hoặc sau w1 w2 chứ k đc đọc giữa chừng.

2.3. Caching:

Thông thường có 3 phương án caching như sau:

- Server caching: dữ liệu thường truy cập sẽ được cache ở phía server (cần cơ chế xử lý cache, vẫn bị độ trễ mạng khi truy cập cached data từ client)

- Client caching: Dữ liệu thường truy cập sẽ được cached ở RAM phía client (truy cập nhanh hơn)

- Client – cache: Cached data dược lưu vào ổ cứng của client. (có thể cache file lớn do ổ cứng lớn hơn RAM. Tuy nhiên truy xuất chậm hơn so với RAM. Có thể truy xuất mà không cần mạng)

Trong distributed sys, cache data có đặc điểm sau:

- Ưu:

* Giảm thời gian truy cập
* Giảm số lượng request (tránh quá tải) cho server nếu data được lưu ở RAM/ ổ cứng của client
* Giảm overhead khi truyền dữ liệu lớn

- Nhược:

* Cần cơ chế đồng bộ (nếu cache data được cập nhật ở server thì cache data ở client sẽ là lỗi thời)
* Nếu write liên tục vào file thì cần đồng bộ liên tục giữa server và cache => tăng overhead time.

=> Cache có lợi khi chủ yếu thực hiện thao tác đọc. Còn chủ yếu là ghi thì cache không có lợi.

2.4. Replication:

- Như đã đề cập ở trên => để tăng tính bảo toàn hệ thống và availabilty => mỗi file được replicated ở trên các nodes khác nhau => cần cơ chế replicate hợp lý và đồng bộ được giữa các replicas.

3. File system của HDFS

- Trong HDFS, data được thiết kế dành cho file lớn. Write once, read many times (nếu viết thêm thì chỉ được append)

- Sử dụng cơ chế đồng bộ UNIX (khi 1 proc viết vào file thì các proc khác ngay lập tức thấy thay đổi.

- File sẽ được chia nhỏ thành các chunk 64 MB => giảm metadata size (giúp lưu ánh xạ chunk đến ca node). Mỗi chunk sẽ được replicated trên 3 nodes khác.

A diagram of data processing

AI-generated content may be incorrect.

Cấu trúc master/slave trong HDFS

- Name node: lưu ánh xạ từ file => chunk lưu file => datanodes lưu các chunk

Trong name node, có các metadata là:

* List các file trong HDFS
* List các chunks cho mỗi file
* List datanodes cho mỗi chunks
* Thông tin về file: thời gian tạo, replication, transaction log của file (thời gian tạo + xóa file)

- Data node: thực sự lưu data thông qua các chunks

* Block server: lưu metadata của chunks, lưu dữ liệu các file thông qua chunks
* Block report: Sẽ thường gửi thông báo trạng thái đến các block trong namenode
* Heartbeat: datanode sẽ gửi hearbeat (trạng thái) của mình đến name nodes mỗi 3s

=> Namenode sẽ dựa vào trạng thái này để xác định tình trạng h/đ mỗi data nodes

- Rack: trong một hệ thống big data, các máy chủ (server) sẽ không để riêng lẻ mà được được xếp vào các rack. Các máy chủ cùng rack với nhau sẽ được kết nối thông qua switch mạng

- Cơ chế replica các chunk trong HDFS:

* Replica đầu tiên của chunk sẽ được lưu ngay trong máy mà tạo ra data đó
* Replica thứ 2: lưu trên một nodes khác nma ở rack khác
* Replica thư 3: lưu trên nodes cùng rack với replica thứ 2

=> Name node cũng đóng vai trò chọn datanode để tạo replica và cân bằng dung lượng giữa các nodes

- Data Rebalancer: sẽ chạy giúp cân bằng lượng data giữa các nodes. Rebalancer thường chạy khi một nodes mới được add vào cụm

- Data corectness: dùng checksum để validate data khi trao đổi giữa HDFS và người dùng. Nếu checksum một chunk sai => chuyển sang replica khác của chunk đó

- Quy trình ghi dữ liệu và replication vào trong data nodes sẽ như sau:

* B1: Client gửi tín hiệu cho namenode để xin danh sách các datanodes để lưu các replicas
* B2: Client gửi chunk tới datanode đầu tiên
* B3: Datanode đầu tiên nhận chunk, gửi các bản sao đến các datanodes kế tiếp
* B4: Khi datanote cuối cùng ghi thành công replica => báo ngược lại đến cho client thông báo ghi thành công

A diagram of a data processing process

AI-generated content may be incorrect.

Pipeline ghi dữ liệu cho HDFS

Name node sẽ lưu metadata ban đầu ở FsImage và lưu lịch sử thay đổi metadata vào Transaction Log (EditLog) => Khi đó, nếu Namenode chạy lâu, Transaction Log sẽ ngày càng nhiều => Mỗi khi khởi động lại namenode thì cần merge lại Transaction Log + FsImage để ra phiên bản hiện tại => tốn thời gian. Chính vì vậy, cần secondary namenode:

* B1: Secondary namenode copy FsImage và EditLog từ Namenode
* B2: Merge FsImage và EditLog để ra phiên bản hiện tại
* B3: Gửi lại FsImage mới lên Namenode
* B4: Xóa EditLog trên namenode => dọn dẹp

=> Thay vì merge khi khởi động namenode, lấy luôn trực tiếp FsImage từ Secondary Image

Một số lệnh thao tác với file trong hdfs:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer file

AI-generated content may be incorrect.

A screenshot of a computer file management

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

4. HDFS file format:  
4.1. File CSV, TSV:

- Data được lưu dưới dạng csv, tsv hoặc json

- Tiện lợi, dễ nhìn với người dùng tuy nhiên không hiệu quả để query

- Không hỗ trợ block compression (để chia ra các chunk)

4.2. Sequence file:

- Lưu dữ liệu dạng cặp key – value

A screen shot of a computer program

AI-generated content may be incorrect.

Cách dữ liệu được lưu trong seq file

=> Do file được lưu dưới dạng key – value => rất thích hợp s/d cho các map reduce job

- Có thể kết hợp nhiều sequence file nhỏ => 1 file lớn (giảm overhead trong metadata)

- Splittable: có thể chia file lớn thành nhiều split để cho nhiều mapper xử lý song song

4.3. Avro:

A screenshot of a computer code

AI-generated content may be incorrect.

Json định dạng các dòng của file

Trong một file Avro sẽ có:

- Header:

* Magic byte = Obj
* Schema = json như trên
* Sync marker

- Data block: là các dòng trong dữ liệu phù hợp hợp với schema đã định nghĩa (không phải ghi lại các key – vd như name đã định nghĩa trong schema) => nhẹ hơn json

4.4. Parquet:

- Thông thường, trong CSV file được lưu dưới dạng hàng. Tuy nhiên, parquet sẽ lưu data dưới dạng cột => khi truy vấn cột sẽ không phải đọc cả file

A black background with white text

AI-generated content may be incorrect.=> A group of white text on a black background

AI-generated content may be incorrect.

Lưu từ CSV => TSV

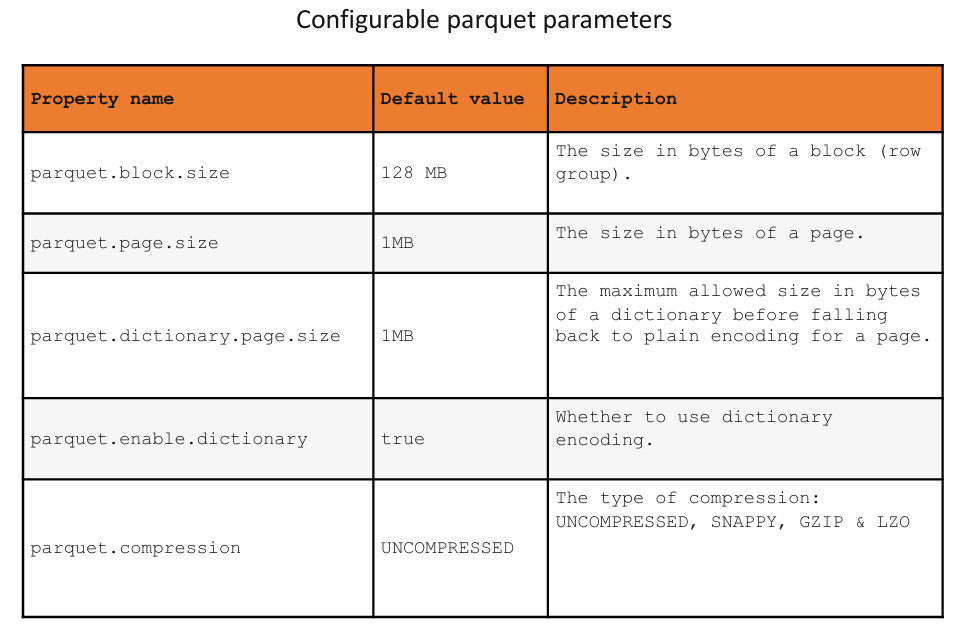
- Do cùng cột thì data có xu hướng format giống nhau => compress dễ hơn CSV

=> S/d parquet khi data hay cần truy vấn 1 or nhiều cột hoặc dữ liệu lớn cần nén

A diagram of a program

AI-generated content may be incorrect.

Cấu trúc của 1 file parquet: => mỗi block ( = row group) lưu một số dòng của file => mỗi column chunk lưu 1 cột của block => mỗi page lưu một số dòng trong cột ở column chunk)



Các tham số có thể điều chỉnh được của file parquet

4.5. OCR:

- Là phiên bản tối ưu hơn của RCFile (định dạng compressed mỗi column riêng biệt)

- Trong OCR, mỗi stripe lưu một số hàng của data (tương tự như row group trong Parquet). Mỗi stripe lưu dữ liệu dưới dạng cột (Giống parquets)

- Điểm khác biệt giữa OCR và Parquet là parquet nén file theo từng page nhưng ocr nén cả stripe

- Mỗi stripe có lightweight index (vd: max, min, average giá trị của từng cột) => có thể bỏ qua các stripe không lq khi truy vấn

- Stripe có thể splittable => phù hợp cho map reduce job hoặc xử lý song song thông qua các mapper

NoSQL

NoSQL: dữ liệu phi cấu trúc

A blue and white table with black text

AI-generated content may be incorrect.

SQL và NoSQL

- NoSQL thường được sử dụng trong hệ thống:

* Yêu cầu khả năng scale cao
* Nhu cầu truy vấn thông tin cao

1. Key/value data model:

- Là loại database lưu trữ dữ liệu dưới dạng cặp key: value. Trong đó:

* Key: định danh duy nhất của dữ liệu
* Value: nội dung dữ liệu

- Ưu điểm:

* Scale rất tốt (do sử dụng cơ chế cơ chế phân tán key, không có quan hệ phức tạp dựa các dữ liệu)
* Khả năng query nhanh

- Nhược điểm:

* Không hỗ trợ query phức tạp: join, group by, filter,…

=> Bản chất trong key/value datamodel, là một table có 2 cột. Cột key đóng vai trò là primary key. Tuy nhiên value có thể ở bất cứ định dạng nào cũng được

- Mem-cache: là một hệ thống cache lưu dữ liệu dạng data – value trên RAM:

* Tốc độ truy cập nhanh (do dữ liệu được lưu ở RAM) => tốc độ cao và giảm workload cho database (thường dùng ở các web hay cần query dữ liệu)
* Chạy distributed server tốt (kết hợp RAM ở các nodes khác nhau)

- Redis: Cũng là hệ csdl nosql lưu RAM. Tuy nhiên có thể lưu dữ liệu vào cả ổ cứng, hỗ trợ nhiều dạng dữ liệu hơn mem cache

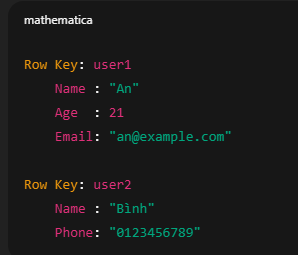
- Amazon DynamoDB: cugnx lưu dữ liệu dạng key, value. Dễ scale, phù hợp với ứng dụng real-time, scale lớn

- Riak: NoSQL dạng key/value. Nổi nhất là khả năng high availability và fault tolerance.

2. Column Family Store:

- Là một loại nosql database lưu trữ dữ liệu theo cột (thay vì theo hàng như sql)

Mỗi row được định danh bằng một row key. Mỗi row key có số lượng cột khác nhau (không cố định schema)



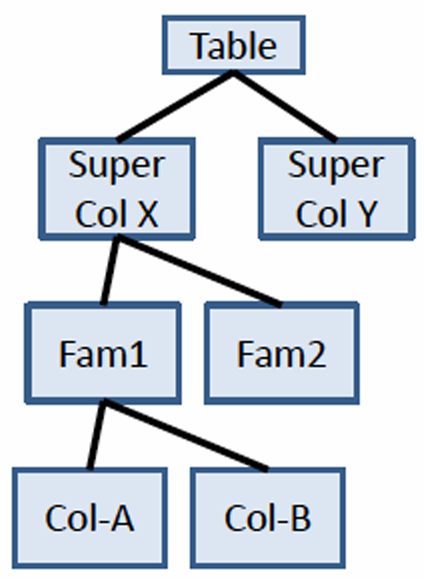
Ví dụ: lưu dữ liệu trong column family store

Uư điểm của column – family store:

* Không cần cố định schema (định dạng dữ liệu ban đầu)
* Uư tiên các truy vấn liên quan đến cột (tính tổng,…)
* Hỗ trợ cơ chế replication => phân tán tốt

Nhược điểm:

* Quản trị sẽ phức tạp hợp so với key/value



Trong col – fam store, các column group lại thành family. Các family group lại thành super column

=> So với relation database (lưu theo row), col-fam store có đặc điểm là có thể thêm column vào trong bảng mà không ảnh hưởng đến các row còn lại trong bảng

- HBASE (hệ quản trị csdl của Hadoop) bản chất lưu data theo col-fam store.

- Apache Cassandra: cũng là một hệ csdl được lưu theo col-fam store

3. Graph data model:

- Lưu data dưới dạng graph. Các nodes liền kề nhau sẽ biết nhau => bản chất query là đi dọc theo graph

- Ví dụ: Hệ quản trị cơ sở dữ liệu Neo4j

A diagram of a algorithm

AI-generated content may be incorrect.

Relational db vs graph

A diagram of a chart

AI-generated content may be incorrect.

Key value vs graph

A diagram of data and a diagram of data

AI-generated content may be incorrect.

Documents vs graph

4. Documents:

- Lưu data dưới dạng 1 file (json hoặc xml) (mỗi file sẽ có 1 id riêng biệt)

5. Object Relational Data Mapping:

- Là một kỹ thuật ánh xạ giữa object và bảng trong csdl quan hệ. Cho phép làm việc với csdl SQL bằng cách thao tác với object => ORM là cầu nối giữa csdl và code

* Bảng => class
* Row => object
* Column => Attribute

- Ưu điểm:

* ORM có thể s/d với mọi hệ qtri csdl khác nhau (MySQL, PostGreSQL)
* Gắn liền với nhiều framework khác nhau
* Tránh trường hợp SQL Injection (hacker s/d các câu lệnh sql độc hại vào database) mà ta có thể lọc trước khi đưa vào database

- Nhược điểm:

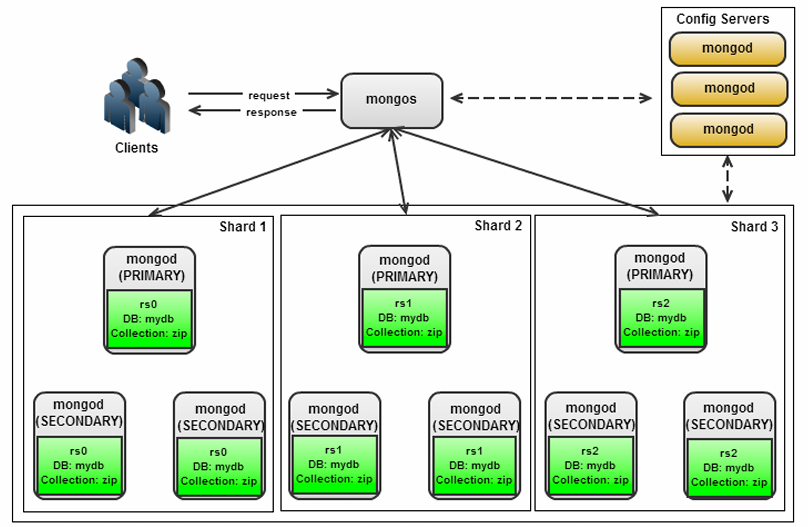
* Có thể chậm hơn SQL thuần
* Truy vấn thông qua ORM thường không nhanh bằng SQL viết tay

6. MongoDB:

- Là 1 hệ quản trị csdl NoSQL dạng Documents

- Là một model dạng master-slave

- MongoDB lưu từng object dưới dạng JSON Document, table được gọi là collection (tập hợp các Document). Để bảo toàn tính fault tolerence, mỗi document trong MongoDB sẽ được nhân bản lên 3 lần. Nhằm tránh quá tải dữ liệu, MongoDB chia dữ liệu thành shard (partition) (một shard có thể chứa 1 or nhiều node). Mỗi shard chứa một số document => khi scale là horizontal scale



Chia dữ liệu trong MongoDB

CAP Theorem

- CAP: viết tắt của 3 tính chất sau

* Consistency: tất cả các clients luôn thấy cùng giá trị của 1 data trong 1 thời điểm (các replica của data phải giống nhau everytime)
* Availability: mỗi client đều có thể read hoặc write vào database ở bất cứ khoảng thời gian nào
* Partition Tolerance: hệ thống tiếp tục hoạt động nếu lỗi mạng khiến các một số nodes không thể giao tiếp được với nhau

A diagram of a pick two

AI-generated content may be incorrect.

CAP Tradeoff và một số hệ qtr csdl phù hợp

=> CAP Theorem: Mỗi hệ thống distributed chỉ đảm bảo được 2 trên 3 tính chất trong CAP Theorem

- Trong hệ relational database, có 2 cách để scale database:

* Vertical scale: tăng hệ thống phần cứng của node => 1 node có thể lưu được nhiều dữ liệu hơn
* Horizontal scale: thêm nhiều node hơn => có thêm node để lưu dữ liệu

- Data sharding: chia data thành nhiều phần nhỏ => có thể xử lý song song. Tuy nhiên, đối với các query phức tạp, yêu cầu truy vấn đến tất cả các shard, thì mất thêm overhead time để truy cập shard, join data,….)

- Data replicating: nhân bản mỗi data thêm các bản sao của nó => giúp hệ thống có fault tolerance, giảm bottleneck (khi có quá nhiều request đến một node cùng lúc). Tuy nhiên data consistency (đồng bộ giữa các replication) cũng là 1 vấn đề

- Two phase commit protocol: giao thức giúp đảm bảo tính nhất quán trong giao dịch giữa nhiều nodes. Có 2 phase chính:

* Phase 1 (Prepare): khi thực hiện một giao dịch giữa 2 nodes khác nhau, bộ điều phối sẽ cần hỏi tính hợp lệ của 2 nodes => nếu node OK thì phản hồi YES, còn đâu là NO
* Phase 2 (Commit): nếu 2 node đều YES thì thực hiện commit, nếu không thì rollback về trạng thái ban đầu

=> Đảm bảo atomic và consistency (trong bộ tính chất ACID)

Ví dụ về CAP Theorem: Khi thiết kế Facebook, cần đảm bảo người dùng có thể đọc data bất cứ khi nào (Consistency) bởi mỗi lần downtimes sẽ thiệt hại rất nhiều tiền. Ngoài ra, do lượng data lớn nên lưu trên nhiều máy chủ => khả năng 1 máy chủ lỗi mạng cao => cần đảm bảo partition tolerance. Chính vì vậy, hệ thống sẽ tradeoff bằng consistency (người dùng khác nhau ở trong một thời điểm có thể thấy data khác nhau)

=> Tùy vào tính chất hệ thống, có thể cân nhắc tradeoff giữa consistency và availability/scability:

* Khi hệ thống yêu cầu dữ liệu luôn chính xác => tăng consistency, giảm availabily và scability
* Khi hệ thống yêu cầu nhanh, dễ mở rộng => giảm consistency, tăng availability và scability

- Base: là một bộ quy tắc trong NoSQL. Nó thiên về availability và scability hơn là strict consistency khi so với ACID trong SQL:

* Basically Availability: hệ thống đạt trạng thái availability => có thể thay đổi/ truy cập data bất cứ lúc nào tuy nhiên dữ liệu chưa chắc mới nhất
* Soft state: kể cả không có request nào, dữ liệu vẫn có khả năng thay đổi để đồng bộ với nhau
* Eventual consistency: nếu không có thay đổi trong một khoảng thời gian => data sẽ đạt trạng thái consistency (all replica same values)

Amazon DynamoDB

- Đặc điểm của hệ thống Amazon DynamoDB:

* Lưu giá trị dưới dạng key/value
* Đánh đổi strong consistency để đổi lấy high availability, Parition Tolerance và High Scability
* Luôn luôn writeable
* Confict resolution sẽ được thực hiện lúc read thay vì write
* Được thiết kế đối xứng (các nodes cùng vai trò), decentralization (giúp tránh được outage (gián đoạn dữ liệu), hegenertority (có tính không đồng nhất, chấp nhận nhiều script or phần cứng khác nhau)

- Giống như các csdl khác, Amazon DynamoDB partition dữ liệu để lưu trên các nodes khác nhau (phục vụ cho cơ chế fault tolerance và scability) => Dynamo partition dữ liệu dựa trên consistent hashing: đưa các node vào một vòng tròn có giá trị 0 => max. Hashing dữ liệu ra value range từ 0 => max. Ta lưu dữ liệu vào nodes gần nhất theo chiều kim đồng hồ => khi thêm or xóa nodes, chỉ 1 phần ít dữ liệu cần di chuyển => good for scability. Amazon là zero – hop DHT, nghĩa là truy vấn trực tiếp đến nodes chứa key mà không phải đi qua hop trung gian.

- Virtual node: là một cơ chế trong Consistent hashing. Một node vật lý sẽ có nhiều virtual nodes được rải rác quay vòng tròn:

* Các virtual ndoes sẽ được ánh xạ trực tiếp đến physical nodes của nó
* Nếu physical nodes hỏng, chỉ cần chia lại virtual nodes của nó cho các virutal ndoes/physical nodes còn lại => dữ liệu vẫn phân tán đều hơn => tăng fault tolerance
* Do physical nodes được rải rác đều trên vòng tròn => tăng load balancing
* Tăng scability do thêm node mới thì chỉ cần chia lại các virtual node (cho 1 số virtual node hiện có trỏ lại vào trong node mới)

- Cơ chế lưu replica: Một key/file K sẽ có N bản sao (N bản sao của nó sẽ được lưu bởi n nodes gần nhất so với nó, tính theo chiều kim đồng hồ)

- Quorum: là số lượng nodes tối thiểu để một thao tác đọc/ghi được coi là hợp lệ => do Dynamo không cần strong consistency, ta không cần tất cả các nodes cùng xác nhận thay đổi. Để đảm bảo consistency, ta cần:

* W+R (số lượng nodes cần xác nhận khi ghi + đọc) > N (số lượng bản sao)

=> sẽ có ít nhất một nodes giao nhau giữa vùng ghi vào đọc => không bị tình trạng đọc ra dữ liệu cũ

=> Dựa vào thay đổi trong Quorum, ta có thể cân bằng giữa availability và consistency

- Cơ chế sloppy quorum (cho temporary failures): khi update a mà một node B lưu replica a bị lỗi => ngay lập tức lưu giá trị a sang node E (node E sẽ lưu thêm dữ liệu này từ B) => khi node B hồi phục thì sẽ trả lại giá trị cho node B => always writeable, fault tolerance.

- Replica synchronization (đồng bộ giữa các replica):

* Merkle tree: là một cấu trúc dữ liệu tree lưu trong một nodes mà leaves là hash values của toản bộ dữ liệu trong nodes. Parent là hash của concat các children. => nếu node con thay đổi, node cha cũng thay đổi
* Khi so sánh để đồng bộ dữ liệu, A sẽ gửi B merkle tree của B => B từ tree lần xuống các nhanh để tìm sự khac biệt (nếu có) => chỉ update nhanh khác biệt (nhánh mà A và B có nhưng giá trị hash khác)

- Véc tơ clock: là một cấu trúc để so sánh version message trả về của một nodes. Quy tắc cập nhật nodes:

* B1: mỗi object (key/files) đều có một véc tơ clock
* B2: nếu A thực hiện update trên file (nó ghi vào véc tơ clock file). Vd: [3,1,0] => [4,1,0]
* B3: mỗi khi trả lại file/key cho người dùng thì cần trả lại vecto clock tương ứng
* B4: các file sẽ merge vector clock trên nodes bằng cách lấy max theo từng vị trí: Vd: A lưu [1,0,0] B lưu [0,1,0] thì merge thành [1,1,0]

=> Người dùng có thể so sánh Véc tơ clock của 2 version của 1 file. Nếu tất cả phần từ ver x lớn hơn ver y => x là phiên bản mới của y => có thể loại bỏ y. Nếu vecto clock 2 ver không so sánh được => 2 sk update song song nhau => cần merge lại.

A diagram of a flowchart

AI-generated content may be incorrect.

Qúa trình cập nhật overclock của 1 file

A white rectangular box with black text

AI-generated content may be incorrect.

Kỹ thuật trong dynamoDB

Apache HBASE

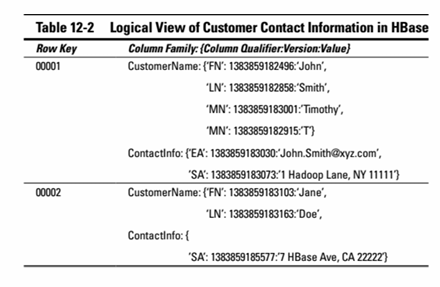
- Là hệ quản trị csdl của Hadoop (muốn truy vấn trực tiếp phải thông qua API hoặc SQL thông qua apache hive)

- Các phần tử trong HBase:

* Column qualifier: Tên cột cụ thể
* Column family: một nhóm cột liên quan đến nhau (sẽ được lưu trên cùng 1 đĩa)
* Row key: khóa chính giúp xác định dòng (row được xếp theo thứ tự từ điển của row key)
* Region: tập hợp liên tiếp các row được sắp xếp
* Timestamp: hỗ trợ versioning (khi query không nói timestamp thì mặc định trả lại version có timestamp gần nhất

=> nên có ít column family

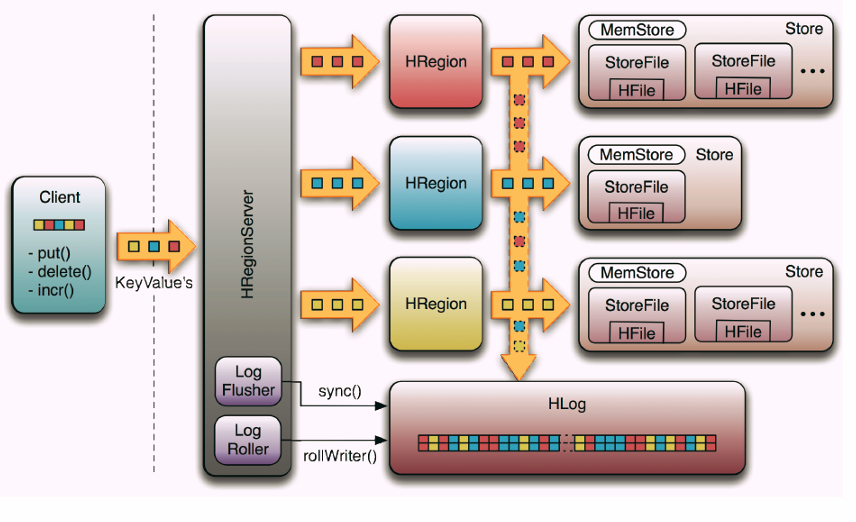
=> 1 value trong 1 cell được x/đ = column, row key và timestamp



Ví dụ về cách cấu trúc dữ liệu trong HBase

- Tính chất của HBase:

* Sparse: Mỗi hàng có thể có số cột khác nhau
* Distributed: dữ liệu được phân tán trên nhiều cluster
* Persistent: Dữ liệu không bị mất khi restart



Flow dữ liệu trong HBase

- HRRegionServer: Quản lý nhiều RegionServer. Mỗi Region Server được để trên 1 node (auto-sharding giúp load balancing giữa các region server), bao gồm nhiều Region Trước khi chuyển dữ liệu cho HRRegion, ta sẽ cần truyền HLog để backup trong trường hợp có server nào crash:

* sync(): flush log xuống đĩa
* rollWriter(): khi log đầy => chuyển sang file log mới

- HRRegion: Mỗi region quản lý nhiều row. Khi dữ liệu đến region, nó sẽ được khi vào memstore, khi memstore đầy thì sẽ flush xuống thành các HFile

- Mỗi store: tương ứng với 1 column family (1 hrregion có thể có nhiều store)

- HFile: đơn vị vật lý lưu trữ dữ liệu trong HBase theo dạng key: value

* Data Block: lưu các cặp dạng key: value (timestamp => value)
* Data Index: cho biết block nào chứa data nào => tìm kiếm nhanh
* Trailer: pointer tới block => Có thể nhảy đến bất cứ phần nào trong file

Do mỗi lần memstore đầy đều flush ra một HFile => nhiều HFile => đọc chậm, quản lý khó => Giải pháp: Compaction:

* Minor Compaction: chạy định kỳ, gộp 1 vài HFile nhỏ thành 1 file lớn
* Major Compaction: gộp tất cả các HFile thành 1 HFile duy nhất

- HMaster: quản lý nhiều HRRegionServer

- Zookeeper: quản lý hoạt động, trạng thái của HMaster, phân bổ RegionServer giữa các HMaster. Còn lưu thêm thông tin của mapping của Row key => Region để khi query người dùng biết cần truy cập HRegion nào lấy data.

Lưu ý: Trong CAP, HBase đảm bảo consistency và partition fault tolerance nhưng không có availability

Về ACID, HBase đảm bảo atomicity và Consistency

=> HBase sẽ phù hợp với những y/c sau:

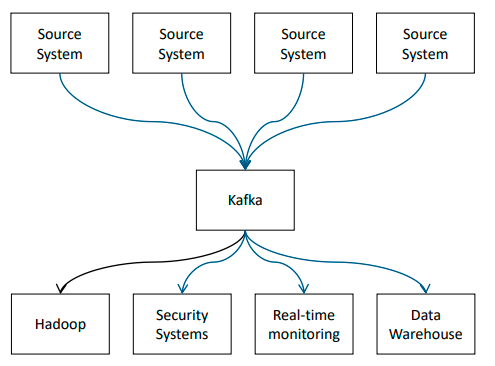
* Cần lookup record nhanh
* Insert column, record nhanh
* Updates data nhiều
* Không join thông tin (nếu join s/d RDBMS)

KAFKA

1. Khái niệm cơ bản về KAFKA:

- KAFKA là một hệ thống truyền dữ liệu, có khả năng mở rộng, bền vững và chịu lỗi.

- KAFKA có thể dùng để tách rời các luồng dữ liệu



KAFKA thường được ứng dụng để truyền dữ liệu dạng real-time vào các storage như hadoop, spark, warehouse.

- Các khái niệm trong KAFKA:

* Topic: message được chia thành các topic tùy vào lĩnh vực (Vd: orders: dữ liệu đơn hàng, users: dữ liệu người dùng)
* Log và Topic storage: Topic chia thành => partition => segment => lưu log trên disk

Topic: orders

├── Partition 0 → segment1.log, segment2.log

├── Partition 1 → segment1.log, segment2.log

* Partition: consumer có thể đọc song song các partition => tăng tốc độ
* Record: Mỗi record gồm: key (xác định vào partition nào) + value (giá trị bản ghi) + timestamp (hỗ trợ versioning)

{

"key": "user123",

"value": {"order\_id": 101, "amount": 250000},

"timestamp": "2025-10-22T10:00:00Z"

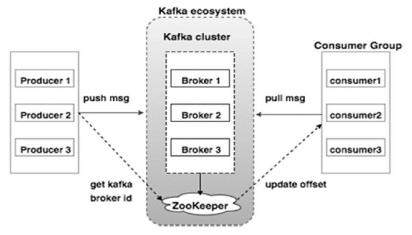
}

Vd: Một record

* Producer: tiến trình gửi dữ liệu vào topic
* Consumer: tiến trình nhận dữ liệu từ topic
* Broker: server lưu dữ liệu để consumer đọc. Có thể có nhiều broker để có tính distributed và fault tolerance

- Zookeeper: giúp quản lý các broker trong cụm kafka

* Quản lý trạng thái các broker
* Lưu metadata: topic, partition, partition nào trong broker nào của kafka
* Chọn ra broker controller (quản lý partition, replica,..)



Zookeeper trong kafka cluster

2. KAFKA Topic architecture:

- Dữ liệu vào KAFKA => topic => partition (dựa vào key)

- Các records lưu trong partition được đánh 1 id gọi là offset

**Chèn ảnh partition by key**

- Cơ chế lưu trong partition:

* Mỗi partition có 1 leader partition hoặc 0 or more followers
* Leader partition được thực hiện chính nhiệm vụ ghi dữ liệu, follower chỉ replicate theo
* Để bảo toàn fault tolerance, các partition leader/followers sẽ được lưu ở broker khác nhau. Khi 1 partition leader die thì 1 thằng followers sẽ được đôn lên làm leader.

Lưu ý: để đảm bảo tính consistency, consumer chỉ được đọc record khi các followers partition đã lưu hết log từ leader partition.

**Chèn ảnh slide 18**