Mục lục

1. Dữ liệu lớn
   1. Định nghĩa
   2. Vấn đề
2. Hadoop
   1. HDFS
   2. YARN
   3. Map reduce
3. Spark

1. Dữ liệu lớn
   1. Định nghĩa

Dữ liệu lớn được định nghĩa là quá trình tăng phi mã của dữ liệu, kể cả ở tổng lượng dữ liệu và tốc độ sinh dữ liệu. Dữ liệu lớn cần những kiến trúc có khả năng mở rộng theo thời gian, đảm bảo được việc lưu trữ, xử lí và phân tích.

* 1. Các đặc điểm của dữ liệu lớn

Đặc điểm của dữ liêụ lớn thể hiện qua các Vs:

* Volume: Tổng lượng dữ liệu lớn
* Velocity: Tốc độ sinh dữ liệu lớn
* Variety: Dữ liệu đa dạng về cấu trúc
* Veracity: Dữ liệu thường thiếu chất lượng
* Value: Dữ liệu lớn chứa những giá trị tiềm ẩn
* Visualization: Dữ liệu cần được biểu diễn để những người không phải kĩ thuật hiểu được.
* Variability: Ý nghĩa của dữ liệu có thể thay đổi theo thời gian và hoàn cảnh.

1. Hadoop
   1. HDFS
2. Giới thiệu chung

HDFS (Hadoop distributed file system) là hệ thống file lưu trữ dữ liệu phân tán.

HDFS được thiết kế phù hợp với những đặc điểm của dữ liệu lớn. Về cơ bản, HDFS có những đặc điểm sau:

* Kích cỡ file lớn: HDFS có thể lưu trữ dữ liệu với lượng lớn, từ gigabyte tới petabyte.
* Ghi 1 lần đọc nhiều lần: HDFS sẽ chi ghi dữ liệu và không thể điều chỉnh dữ liệu được ghi. Điều này giúp tăng tốc độ ghi dữ liệu vì chỉ cần ghi vào cuối.
* Giá rẻ: HDFS có thể được triển khai trên các phần cứng giá rẻ
* Khả năng mở rộng: HDFS có thể được mở rộng bằng cách thêm datanode
* HDFS sẽ chia dữ liệu thành các block lớn. Điều này giúp cho việc quản lí dữ liệu hiệu quả hơn.
* Khả năng chịu lỗi: HDFS sẽ lưu trữ các block dữ liệu thành nhiều bản sao và phân tán trên các máy khác nhau. Khi một máy chết thì dữ liệu của máy đó vẫn sẽ được đảm bảo trên các máy khác.

Các thành phần chính của HDFS bao gồm: Name node, Data node.

1. Data node

Các data node là nơi chứa dữ liệu thật của HDFS. File lưu dữ liệu trên data node được cấu hình bởi *dfs.datanode.data.dir* trong file *hdfs-site.xml.*

1. Name node

Name node chứa cấu trúc cây thư mục, metadata của tất cả file, block dữ liệu của HDFS.

Thông tin metadata được lưu tại name node thông qua 2 đối tượng:

* Namespace Image File (FS Image): Là một điểm checkpoint của HDFS. File chứa FS Image được cấu hình bởi *dfs.namenode.name.dir* trong *hdfs-site.xml*
* Edit Log: Là 1 nhật kí chứa tất cả các thay đổi của HDFS kể từ check point cuối cùng. File chứa Edit log được cấu hình bởi *dfs.namenode.edits.dir* trong *hdfs-site.xml*

Khi khởi động Name node, Name node đầu tiên sẽ đọc file FS Image để lấy check point, sau đó đọc Edit log và thực hiện tất cả các thay đổi từ lúc check point để khôi phục trạng thái HDFS.

Secondary Name node sẽ đảm nhiệm vai trò cập nhật FS image và Edit log để tạo thành các check point (FS Image) mới.

Khi client ghi thêm dữ liệu mới vào HDFS, name node sẽ tìm các datanode để ghi dữ liệu và client sẽ ghi dữ liệu vào trực tiếp trên các datanode đó (kể cả các bản relication). Các datanode sẽ gửi lại địa chỉ lưu trữ block cho name node để lưu vào meta data.

Khi client muốn đọc dữ liệu, name node sẽ tìm các data node khả dụng chứa block của dữ liệu cần đọc. Sau đó, client sẽ trực tiếp đọc dữ liệu trên các data node.

Trong cấu trúc High Avaibility (HA) của HDFS, NameNode sẽ được backup bằng Standby NameNode. Node này sẽ trở thành NameNode của cụm khi NameNode chết. Để đảm bảo trạng thái của Standby NameNode luôn sẵn sàng để thay thế NameNode bất cứ khi nào, sẽ có các Journal Node làm nhiệm vụ ghi các log về sự thay đổi metadata trên namenode. Standby Node sẽ làm nhiệm vụ checkpointing khi kết hợp các bản log của Journal Node và bản FS Image của mình. Số lượng Journal Node là số lẻ và NameNode sẽ chỉ thay đổi metadata khi đảm bảo đa số các Journal Node đã cập nhật log thay đổi.

Distcp là cách mà Hadoop copy dữ liệu từ một cluster này sang một cluser khác.

* 1. **Yarn**

Yarn được coi là hệ điều hành cho một cụm Hadoop. Yarn sẽ quản lí tài nguyên của cụm và phân bổ tài nguyên cho các job trên cụm.

A screenshot of a computer screen

Description automatically generated

Yarn có 2 thành phần chính:

* **Resource Manager**: là trung tâm điều phối và quản lý tài nguyên toàn cluster.
* **Node Manager**: là một thành phần chạy trên mỗi nút trong cluster và là trách nhiệm quản lý tài nguyên và tác vụ trên từng máy chủ cụ thể đó.

1. **Resource Manager:**

ResourceManager có các vai trò và trách nhiệm chính sau:

* **Quản lý tài nguyên cluster**: ResourceManager giữ vai trò là điểm kiểm soát chính để phân bổ tài nguyên giữa tất cả các ứng dụng (tính toán) trong cluster. Nó theo dõi sự sẵn có của tài nguyên máy chủ (CPU, bộ nhớ, v.v.) và quyết định cách phân phối chúng giữa các ứng dụng dựa trên các yêu cầu và chính sách ưu tiên.
* **Lập lịch (Scheduler)**: Scheduler dựa trên cấu hình chính sách (như FIFO, CapacityScheduler, FairScheduler) để quyết định ứng dụng nào nhận tài nguyên tiếp theo.
* **ApplicationMaster:** Khi một ứng dụng được gửi đến YARN, ResourceManager gán một container để khởi động ApplicationMaster của ứng dụng đó, qua đó quản lý vòng đời và việc thực thi tác vụ của ứng dụng.

1. **Node Manager**

NodeManager là một thành phần chạy trên mỗi nút trong cluster và là trách nhiệm quản lý tài nguyên và tác vụ trên từng máy chủ cụ thể đó:

* **Quản lý Container**: NodeManager quản lý các container, bao gồm việc khởi tạo, thực thi, và giám sát tác vụ ứng dụng trong các container theo yêu cầu từ ResourceManager và ApplicationMaster. Container là các đơn vị tài nguyên logic được cấp phát cho mỗi tác vụ.
* **Giám sát tài nguyên**: NodeManager theo dõi và báo cáo việc sử dụng tài nguyên (CPU, bộ nhớ, đĩa, mạng) trên nút của mình cho ResourceManager, giúp RM đưa ra quyết định phân bổ tài nguyên thông tin.
  1. **Map Reduce**

MapReduce là một mô hình lập trình được thiết kế để xử lý và phân tích dữ liệu lớn một cách dễ dàng trên các cluster máy tính.

Một số bước của Map Reduce:

* **Map:** Mỗi tác vụ map xử lý một phần của tài liệu văn bản và tạo ra các cặp khóa/giá trị
* **Shuffle and Sort:** Hệ thống Hadoop tự động "shuffle" và sắp xếp các cặp khóa/giá trị từ bước Map, nhóm chúng dựa trên khóa (từ)
* **Reduce:** Mỗi tác vụ reduce nhận một nhóm cặp khóa/giá trị và tổng hợp chúng

Ví dụ: (Word Count) Ta có 1 cụm Hadoop lưu trữ văn bản. Ta muốn tổng hợp xem số lượng mỗi từ xuất hiện trên các văn bản.

Giả sử ta có 3 văn bản sau: “Hello Hadoop, welcome to the world of big data.”, “Big data is changing the world.” và “Hadoop is part of the big data ecosystem.”. Mỗi văn bản này lưu trên một máy trong cụm.

1. Map

Mỗi dòng của văn bản được xử lý bởi một tác vụ Map khác nhau, tạo ra các cặp khóa/giá trị như sau (lưu ý rằng chúng ta sẽ chuyển tất cả chữ cái về dạng chữ thường để đếm không phân biệt chữ hoa/chữ thường):

Văn bản 1: "Hello Hadoop, welcome to the world of big data."

(hello, 1), (hadoop, 1), (welcome, 1), (to, 1), (the, 1), (world, 1), (of, 1), (big, 1), (data, 1)

Văn bản 2: "Big data is changing the world."

(big, 1),(data, 1), (is, 1), (changing, 1), (the, 1), (world, 1)

Văn bản 3: "Hadoop is part of the big data ecosystem."

(hadoop, 1), (is, 1), (part, 1), (of, 1), (the, 1), (big, 1), (data, 1), (ecosystem, 1)

2. Shuffle and Sort

Hadoop tự động "shuffle" và sắp xếp các cặp khóa/giá trị từ các bước Map, nhóm chúng dựa trên khóa:

(big, [1, 1, 1])

(changing, [1])

(data, [1, 1, 1])

(ecosystem, [1])

(hadoop, [1, 1])

(hello, [1])

(is, [1, 1])

(of, [1, 1])

(part, [1])

(the, [1, 1, 1])

(to, [1])

(welcome, [1])

(world, [1, 1])

3. Reduce Step

Cuối cùng, mỗi nhóm từ bước trước được xử lý bởi tác vụ Reduce để tính tổng số lần xuất hiện của từng từ:

(big, 3)

(changing, 1)

(data, 3)

(ecosystem, 1)

(hadoop, 2)

(hello, 1)

(is, 2)

(of, 2)

(part, 1)

(the, 3)

(to, 1)

(welcome, 1)

(world, 2)

1. **Spark**

Giống với Map Reduce, Spark là một công cụ để xử lí dữ liệu trên môi trường phần tán.

A diagram of a driver act as executor

Description automatically generated

Spark bao gồm Driver và các Executor.

Khi submit Spark job lên Yarn, quá trình hoạt động diễn ra như sau:

* Khởi tạo và Submit Job:

Khi submit job, SparkContext trong driver program sẽ tạo ra một Spark application. SparkContext sẽ liên lạc với Yarn ResourceManager để yêu cầu khởi tạo một ApplicationMaster cho ứng dụng Spark. ResourceManager sẽ tìm kiếm NodeManager có sẵn và khởi động ApplicationMaster trên một trong những NodeManager đó.

* Khởi tạo ApplicationMaster:

ApplicationMaster là một quy trình quản lý việc thực thi của ứng dụng Spark trên Yarn. Nó sẽ yêu cầu ResourceManager cung cấp tài nguyên (ví dụ: containers) để khởi động các executor của Spark. Mỗi executor được khởi động trong một hoặc một số container Yarn, được quản lý bởi NodeManager.

* Thực thi Job:

Khi các executor đã sẵn sàng, driver sẽ phân chia công việc thành các tasks và gửi chúng đến các executor để thực thi. Các executor sẽ bắt đầu thực thi các task đồng thời. Trong quá trình này, executor có thể liên lạc với HDFS hoặc các hệ thống lưu trữ khác để đọc hoặc ghi dữ liệu.

* Thu thập kết quả:

Khi các tasks hoàn thành, kết quả được thu thập lại và gửi về cho driver program.Driver có thể thực hiện thêm các phép toán trên kết quả hoặc ghi kết quả xuống hệ thống lưu trữ.

* Kết thúc Job:

Khi công việc hoàn tất, ApplicationMaster sẽ thông báo cho ResourceManager để giải phóng tài nguyên và đóng ứng dụng. Driver program sẽ kết thúc, và SparkContext sẽ đóng lại.

**3.1. Spark API**

1. **RDDs**

RDD, viết tắt của Resilient Distributed Dataset, là một trong những khái niệm cốt lõi và là cấu trúc dữ liệu chính trong Apache Spark. RDD đại diện cho một tập hợp dữ liệu phân tán (Distributed), bất biến (immutable), có khả năng phục hồi tự động (Resilient) từ các lỗi, và có thể được xử lý trên nhiều nút của cụm để thực hiện xử lý dữ liệu song song.

* Bất biến: Một khi được tạo, RDDs sẽ không thay đổi. Thực hiện các phép biển đổi trên RDD thực chất là tạo ra 1 RDD mới.
* Tính chống lỗi; RDD sẽ tự động phục hồi dựa trên đặc điểm của lineage (phả hệ). Lineage là việc Spark Driver sẽ ghi nhớ những phép biến đổi dẫn tới RDD để có thể khôi phục khi xảy ra lỗi.
* Phân tán: RDD được chia làm các partition. Các partition này sẽ được xử lí song song trên các máy.
* Có thể lưu một RDD vào memory hoặc disk. Điều này làm tăng tốc độ tính toán khi cần sử dụng lặp lại RDD
* Lazy evaluate: Spark sẽ không vội thực hiện ngay các biến đổi trên RDD mà nó lưu các biến đổi (lineage). Chỉ khi có các lệnh cần in ra như ‘show()’ , ‘count()’, ‘saveAsText()’,… thì phép biến đổi mới thực sự xảy ra.

RDDs là low-API trong Spark. Về cơ bản, ít trường hợp dùng RDDs

1. **DataFrames**

Là API phổ biến nhất trong Spark. DataFrame được xây dựng trên RDDs. Dưới đây là các đặc điểm của DataFrames:

* Các đặc điểm như bất biến, khả năng chịu lỗi, phân tán, caching, lazy evaluate đều giống với RDD
* DataFrame có thêm các hàng và các cột , giống như 1 DataSheet. Schema là kiểu dữ liệu của các cột. Schema có thể do người dùng định nghĩa hoặc Spark có thể tự định nghĩa bằng cách đọc một số dòng. Mỗi hàng sẽ là đại diện cho 1 kiểu dữ liệu
* Tích hợp với dữ liệu đa dạng: DataFrame hỗ trợ đọc và ghi dữ liệu từ và đến nhiều định dạng và hệ thống lưu trữ khác nhau, bao gồm các file cấu trúc như JSON, CSV, Parquet, Avro, và các nguồn dữ liệu như JDBC, Tableau, HDFS, S3, và nhiều hệ thống lưu trữ khác.
* Vì dữ liệu trong DataFrame được tổ chức thành dạng bảng giống với cơ sở dữ liệu truyền thống, DataFrame tích hợp với SparkSql, giúp thực hiện các câu lệnh như đang viết một câu query SQL.

1. **Dataset**

Khá tương đồng với DataFrame.

Khác biệt chính của Dataset với DataFrame là DataSet sẽ check kiểu dữ liệu của cột tại compile time trong khi DataFrame sẽ check kiểu dữ liệu của cột tại run time.