Dokumen ini merupakan materi tambahan mata kuliah Infrastruktur Big Data digunakan untuk melengkapi materi yang sudah diberikan ketika tatap muka.

Apache Cassandra

Apache Cassandra adalah sistem manajemen basis data NoSQL yang dirancang untuk menangani pengolahan dan penyimpanan data yang besar dan terdistribusi dengan cara yang efisien dan scalable. Berikut adalah beberapa poin penting mengenai Apache Cassandra:

- Distributive dan Skalabilitas: Cassandra dirancang untuk menjalankan sistem basis data terdistribusi, yang berarti bahwa data dapat disimpan di beberapa server (node) tanpa mengurangi ketersediaan dan performa. Ini membuat Cassandra sangat sesuai untuk aplikasi yang memerlukan ketersediaan tinggi dan dapat diskalakan secara horizontal.
- Penanganan Data Besar: Sistem ini sangat cocok untuk aplikasi yang menghasilkan atau memerlukan akses ke data dalam volume besar, seperti aplikasi analitik, real-time big data, dan Internet of Things (IoT).
- Keandalan dan Toleransi Kesalahan: Cassandra memiliki mekanisme replika data di mana data disalin di beberapa node. Ini memastikan bahwa jika satu node gagal, data masih dapat diakses dari node lain, memberikan ketahanan terhadap kesalahan.
- Model Data: Cassandra menggunakan model penyimpanan berbasis kolom, alih-alih baris seperti yang umum di basis data relasional. Data diorganisir dalam tabel dengan kolom yang dapat memiliki tipe data yang berbeda.
- Cassandra Query Language (CQL): Cassandra menyediakan bahasa kueri yang mirip SQL yang dinamakan CQL, yang memudahkan pengguna dalam melakukan operasi CRUD (Create, Read, Update, Delete) pada data, meskipun tidak memiliki semua fitur yang ada dalam SQL karena sifat NoSQL-nya.
- Gossip Protocol: Cassandra menggunakan protokol gossip untuk komunikasi antar node dalam cluster, yang memungkinkan node untuk saling berbagi informasi mengenai status sistem dan status kesehatan node lainnya.
- Use Cases: Cassandra banyak digunakan dalam berbagai industri, termasuk perbankan, periklanan, media sosial, dan e-commerce, untuk aplikasi yang membutuhkan kecepatan, skalabilitas, dan ketersediaan tinggi.

Unit Dasar dalam Apache Cassandra

1. Node:
    - Unit dasar dari cluster Cassandra. Setiap node adalah instance dari Cassandra yang berdiri sendiri, menyimpan data, dan memproses permintaan.

2. Cluster:
    - Sekelompok node yang bekerja bersama untuk menyimpan dan mengelola data. Cluster memberikan skalabilitas dan ketersediaan tinggi dengan mendistribusikan data di antara banyak node.
3. Data Center:
    - Sekumpulan node dalam cluster yang berada dalam lokasi fisik yang sama. Data center memungkinkan distribusi data di lokasi geografis berbeda dan meningkatkan pemulihan bencana serta mengurangi latensi akses.
4. Rack:
    - Dalam konteks data center, rack adalah grup dari node yang sering kali terpisah secara fisik dalam infrastruktur server. Penggunaan informasi ini dapat meningkatkan redundansi dan efisiensi penyimpanan data.
5. Table:
    - Struktur data di dalam Cassandra yang menyimpan data dalam format baris dan kolom. Tabel memiliki skema yang mendefinisikan kolom-kolom dan tipe data yang ada.
6. Partition:
    - Unit penyimpanan dasar dalam tabel yang menyimpan baris-baris data. Kunci partisi digunakan untuk menentukan di mana data akan disimpan di seluruh cluster dan memastikan distribusi yang merata.
7. Column Family:
    - Cara untuk mengelompokkan kolom dalam tabel. Column family digunakan untuk mengorganisir data pada level yang lebih tinggi, dengan setiap column family memiliki set kolom yang berbeda.
8. Keyspace:
    - Pangkalan data di Cassandra yang berisi satu atau lebih tabel. Keyspace mengelompokkan tabel yang memiliki pengaturan yang sama, termasuk parameter replika.
9. Replica:
    - Salinan data yang disimpan di node lain dalam cluster untuk memastikan ketersediaan dan keandalan. Jumlah replika ditentukan oleh replication factor namun pada Cassandra jumlah replication factor standarnya adalah 3.
10. Secondary Index:
    - Indeks tambahan yang dapat dibuat di satu atau lebih kolom non-prime untuk meningkatkan efisiensi pencarian data.

Collection Types

1. List

- Deskripsi: Koleksi terurut yang dapat menyimpan elemen duplikat. Pengguna dapat mengakses elemen berdasarkan indeksnya.
- Karakteristik:
    - Elemen order penting dan dapat diakses dengan index.
    - Memungkinkan nilai duplikat.

2. Set

- Deskripsi: Koleksi yang menyimpan elemen unik dan tidak terurut. Set tidak memperbolehkan duplikasi dan tidak memiliki urutan pada elemen.
- Karakteristik:
  - Menjaga keunikan elemen.
  - Pengguna tidak dapat menentukan urutan elemen.

3. Map
   - Deskripsi: Koleksi pasangan kunci-nilai, di mana setiap kunci unik dan digunakan untuk mengakses nilai yang terkait. Map mirip dengan dictionary dalam berbagai bahasa pemrograman.
   - Karakteristik:
     - Memungkinkan penyimpanan data yang terkait dengan kunci.
     - Kunci harus unik dalam map.
4. Tuple
   - Deskripsi: Koleksi terurut dari elemen yang dapat memiliki tipe data berbeda. Tuple memungkinkan pengguna untuk menggabungkan berbagai tipe data dalam satu entitas.
   - Karakteristik:
     - Elemen tersimpan dalam urutan dan dapat memiliki tipe yang berbeda.
     - Berguna untuk menyimpan struktur data yang kompleks.

Snitch

Snitch dalam Apache Cassandra adalah komponen yang berfungsi untuk memberikan informasi tentang topologi jaringan dan lokasi fisik dari node-node dalam cluster Cassandra. Snitch memainkan peran penting dalam pengalokasian data, pemilihan node untuk penulisan dan pembacaan data, serta pengelolaan replikasi data.

Fungsi Utama Snitch

- Pemetaan Lokasi: Snitch mengidentifikasi lokasi fisik dari node (baik dalam konteks data center dan rack) dalam infrastruktur jaringan. Ini memungkinkan Cassandra untuk mengetahui di mana data disimpan dan membuat keputusan cerdas saat mendistribusikan data.
- Pembagian Beban: Dengan mengetahui lokasi node, Cassandra dapat membagi beban kerja lebih merata di seluruh node, mengoptimalkan latensi, dan meningkatkan performa.

Tipe Data CQL (Cassandra Query Language)

1. Tipe Data Numerik

- int: Menyimpan bilangan bulat 32-bit.
- bigint: Menyimpan bilangan bulat 64-bit.
- smallint: Menyimpan bilangan bulat 16-bit.

- tinyint: Menyimpan bilangan bulat 8-bit.
- float: Menyimpan angka floating point 32-bit.
- double: Menyimpan angka floating point 64-bit.
- decimal: Menyimpan angka desimal dengan presisi tinggi (berbasis pada tipe data BigDecimal).

2. Tipe Data Teks

- text: Menyimpan string teks tidak terbatas.
- varchar: Mirip dengan `text`, untuk menyimpan string dengan panjang variabel.
- char: Menyimpan string dengan panjang tetap (1 karakter).

3. Tipe Data Boolean

- boolean: Menyimpan nilai benar (true) atau salah (false).

4. Tipe Data Waktu dan Tanggal

- timestamp: Menyimpan tanggal dan waktu dalam format timestamp (UTC).
- date: Menyimpan hanya tanggal (tanpa waktu) dalam format tahun, bulan, dan hari.
- time: Menyimpan waktu (tanpa tanggal) dalam format jam, menit, detik, dan nanodetik.
- timeuuid: UUID yang dihasilkan berdasarkan waktu. Berguna untuk membuat identifier unik yang juga menyimpan data waktu.

5. Tipe Data UUID (Universally Unique Identifier)

- uuid: Menyimpan UUID, yang merupakan identifier unik yang sangat berguna dalam distribusi data.

6. Tipe Data Collection

- list: Menyimpan koleksi terurut dari elemen, yang dapat memiliki duplikat.
- set: Menyimpan kumpulan elemen unik tanpa urutan.
- map: Menyimpan pasangan kunci-nilai, di mana setiap kunci unik.

7. Tipe Data Tuple

- tuple: Koleksi terurut dari elemen yang dapat memiliki tipe data yang berbeda. Berguna untuk menyimpan struktur data yang kompleks.

8. Tipe Data Blob

- blob: Menyimpan data biner, yang dapat digunakan untuk menyimpan file atau informasi dalam format tidak terstruktur.

9. Tipe Data Custom

- Cassandra juga mendukung tipe data kustom yang dapat dibuat oleh pengguna (user-defined types / UDT). Ini memungkinkan pengguna untuk mendefinisikan tipe data yang kompleks dengan atribut yang relevan.

Susunan Direktori Cassandra

1. Direktori Utama

- bin/:
    - Berisi skrip eksekusi Cassandra, termasuk scripts untuk menjalankan dan mengelola node Cassandra seperti `cassandra`, `cqlsh`, dan `nodetool`.

- conf/:
    - Menyimpan file konfigurasi Cassandra. File utama adalah `cassandra.yaml`, yang berisi pengaturan konfigurasi dasar untuk cluster, node, dan aspek lainnya seperti pengaturan jaringan, pemulihan, dan pengaturan snitch.
- data/:
    - Lokasi di mana semua file data Cassandra disimpan. Terdapat subfolder untuk setiap keyspace yang dibentuk di dalam cluster. Di dalam keyspace, data diorganisir oleh tabel dan disimpan dalam format SSTable.
- commitlog/:
    - Berisi commit log yang menyimpan semua operasi penulisan yang sedang dilakukan. Commit log digunakan untuk menjaga integritas data dan memulihkan data dalam kasus kegagalan.
- saved_caches/:
    - Menyimpan cache yang diperlukan untuk meningkatkan performa database dengan mengurangi kebutuhan untuk membaca data dari disk.
- hints/:
    - Menyimpan hint, yaitu informasi yang digunakan untuk membantu dalam pemulihan data yang gagal ditulis ke node yang tidak tersedia saat penulisan.

2. Direktori Tambahan

- lib/:
    - Berisi library Java yang diperlukan oleh Cassandra untuk beroperasi. Ini termasuk driver dan dependensi lainnya.
- logs/:
    - Menyimpan file log untuk berbagai proses dalam Cassandra, termasuk log untuk debug, warning, dan informasi lainnya mengenai status operasional node.
- tools/:
    - Menyediakan tambahan alat untuk administrasi dan pengelolaan cluster Cassandra, termasuk alat pengujian dan monitoring.

Apache Spark

Apache Spark adalah sistem komputasi terdistribusi yang dirancang untuk pemrosesan data besar (big data) dengan kecepatan tinggi. Spark menawarkan berbagai alat dan pustaka untuk analisis dan pemrosesan data, baik dalam bentuk terstruktur maupun tidak terstruktur. Berikut adalah beberapa fitur dan karakteristik utama dari Apache Spark:

1. Kecepatan Tinggi: Spark dapat memproses data dalam memori (in-memory computing), yang membuatnya lebih cepat dibandingkan dengan sistem pemrosesan data tradisional seperti Hadoop MapReduce.
2. Kompatibilitas: Spark dapat berjalan di atas infrastruktur Hadoop, dan dapat menggunakan Hadoop Distributed File System (HDFS) untuk menyimpan data.
3. API yang Fleksibel: Spark menyediakan API yang mendukung beberapa bahasa pemrograman, termasuk Scala, Java, Python, dan R. Ini memungkinkan pengembang untuk menggunakan alat yang paling nyaman bagi mereka.
4. Komponen Utama:
    - Spark SQL: Untuk pemrosesan data terstruktur menggunakan SQL.
    - Spark Streaming: Untuk pemrosesan data streaming secara real-time.
    - MLlib: Untuk pengembangan dan penerapan algoritma machine learning.
    - GraphX: Untuk pemrosesan graf dan analisis.
5. Pemrosesan Batch dan Stream: Spark mendukung pemrosesan batch (data yang diolah dalam kelompok) serta pemrosesan data streaming (data yang diproses secara real-time).
6. Ekosistem yang Luas: Spark memiliki ekosistem yang mendukung berbagai alat dan pustaka yang dapat digunakan bersama dengannya, termasuk Apache Hive, Apache HBase, dan Apache Kafka.

Komponen Arsitektur Spark

1. Manajer Cluster

Manajer cluster adalah komponen yang mengelola dan mengatur sumber daya di dalam cluster. Tugas utamanya mencakup:

- Alokasi Sumber Daya: Memastikan bahwa sumber daya seperti CPU dan memori tersedia dan dialokasikan di antara berbagai aplikasi yang berjalan di cluster.
- Pengelolaan Proses Executor: Menjaga proses executor yang menjalankan tugas pemrosesan data, baik dalam mode jarak jauh maupun dalam berbagai mode lain yang didukung oleh Spark.
- Penyewaan dan Penjadwalan: Menyewa dan menjadwalkan tugas untuk eksekusi, memastikan pemanfaatan sumber daya yang optimal di seluruh cluster.

2. Driver Program

- Pengelola Aplikasi: Driver adalah program utama yang melakukan kontrol eksekusi aplikasi Spark, termasuk mengelola tugas dan executor.

- Komunikasi: Berfungsi sebagai otak dari aplikasi, berkomunikasi dengan cluster manager dan executor.

3. Executor

- Proses Penjalankan Tugas: Executor adalah proses yang berjalan di setiap node dalam cluster. Mereka bertanggung jawab untuk menjalankan tugas yang ditempatkan oleh driver dan menyimpan data sementara yang dibutuhkan oleh aplikasi.
- Manajemen Sumber Daya: Setiap executor mengelola sumber daya yang dialokasikan oleh manajer cluster untuk menyelesaikan pekerjaan yang diberikan.

4. Task

- Unit Pekerjaan Terkecil: Task adalah unit terkecil dari pekerjaan yang dieksekusi dalam Spark. Setiap task mewakili sebuah pekerjaan yang dipecah dari operasi Spark yang lebih besar.
- Distribusi Eksekusi: Task didistribusikan oleh driver ke executor untuk dilaksanakan secara paralel.

5. Data Sources dan Sink

- Sumber Data: Spark dapat bekerja dengan berbagai sumber data seperti HDFS, Apache Kafka, Apache Cassandra, dan lain-lain.
- Tempat Penyimpanan Hasil: Hasil pemrosesan bisa disimpan kembali ke berbagai format dan sistem penyimpanan untuk analisis lebih lanjut.

6. Cluster Driver

- Pengaturan Tugas: Cluster driver bertanggung jawab untuk mengatur dan memantau eksekusi aplikasi Spark, mengelola penyebaran tugas ke executor.
- Interaksi dengan Manajer Cluster: Berkomunikasi dengan manajer cluster untuk memesan sumber daya yang diperlukan.

7. API dan SDK

- Bahasa Pemrograman: Spark menyediakan API dalam berbagai bahasa pemrograman seperti Scala, Java, Python, dan R, memungkinkan pengembang untuk menggunakan alat yang mereka pilih untuk pengembangan aplikasi.

Spark Execution Hierarchy

Spark execution hierarchy adalah struktur organisasi yang digunakan oleh Apache Spark untuk merencanakan, mengelola, dan mengeksekusi pemrosesan data di dalam sistem terdistribusi. Hierarki ini memberikan gambaran tentang bagaimana pekerjaan dibagi menjadi unit-unit yang lebih kecil dan bagaimana unit-unit tersebut dieksekusi oleh cluster. Berikut adalah komponen utama dari Spark execution hierarchy:

1. Application

- Ini adalah entitas tertinggi dalam hierarki. Sebuah aplikasi Spark terdiri dari kode yang dituliskan oleh pengembang yang melakukan pemrosesan data. Aplikasi ini dapat mencakup berbagai operasi, seperti transformasi dan tindakan pada dataset.

2. Job

- Setiap aplikasi dapat memiliki satu atau lebih job. Job dipecah dari operasi tindakan (action) yang memicu pemrosesan data. Misalnya, jika ada tindakan yang memerlukan hasil akhir (seperti `count()` atau `collect()`), itu akan memicu job baru.

3. Stage

- Job dibagi menjadi beberapa stage, yang merupakan kelompok dari satu atau lebih tugas (task). Stages biasanya dibedakan berdasarkan dependensi antara operasi. Jika satu operasi bergantung pada hasil dari operasi lain, mereka akan dikelompokkan dalam stage yang berbeda.
- Pemisahan Stages: Stage dibagi dalam dua kategori:
    - Narrow Dependencies: Operasi yang dapat dieksekusi dalam satu stage karena setiap partisi data dari operasi sebelumnya bisa digunakan oleh satu partisi data dari operasi berikutnya.
    - Wide Dependencies: Operasi yang menyebabkan shuffle — data dari beberapa partisi perlu dipindahkan ke partisi lain, mengakibatkan pembagian ke dalam stage yang berbeda.

4. Task

- Task adalah unit terkecil dalam hierarki dan merupakan bagian dari stage. Setiap task dieksekusi pada partisi data tertentu dan dapat dijalankan secara paralel oleh executor.
- Eksekusi: Setiap task menjalankan pekerjaan pada partisi dataset dan biasanya berfungsi sebagai unit yang mengelola data yang dieksekusi berdasarkan rencana yang ditetapkan oleh driver.

Alur Eksekusi

Proses eksekusi diawali ketika pengembang menjalankan aplikasi Spark:

1. Aplikasi Spark dimulai.
2. Ketika tindakan (action) dipanggil, job akan dibuat.
3. Job akan dipecah menjadi stages berdasarkan dependensi dan kebutuhan shuffle.
4. Setiap stage terdiri dari beberapa tasks, yang masing-masing dieksekusi pada partition data yang berbeda oleh executor.

Dynamic Partition Pruning

Dynamic partition pruning adalah strategi yang memungkinkan sistem untuk memotong atau mengabaikan partisi yang tidak diperlukan dalam hasil kueri berdasarkan kondisi filter yang diterapkan saat menjalankan kueri. Ini dilakukan secara dinamis selama eksekusi kueri, bukan saat kompilasi rencana kueri.

Manfaat Dynamic Partition Pruning

1. Efisiensi Waktu: Mengurangi jumlah partisi yang perlu diproses dapat mengurangi waktu eksekusi kueri secara signifikan.

2. Penghematan Sumber Daya: Dengan tidak memproses data yang tidak relevan, lebih sedikit sumber daya seperti CPU dan memori yang digunakan, yang pada gilirannya mengurangi biaya operasional.
3. Peningkatan Performa: Dynamic partition pruning dapat membuat kueri lebih responsif dan meningkatkan performa keseluruhan sistem, terutama pada dataset yang sangat besar.

Perbandingan Spark dan Hadoop

Aspek	Apache Spark	Hadoop
Model Pemrosesan	In-memory computing (memproses data di memori)	Disk-based processing (mengandalkan operasi I/O disk)
Kinerja	Lebih cepat dalam pemrosesan data, terutama untuk iterasi	Lebih lambat karena menuntut lebih banyak operasi disk
Kemudahan Penggunaan	API yang lebih intuitif (DataFrame, RDD, dan SQL)	API kompleks dan lebih sulit untuk pengguna baru
Tipe Data yang Didukung	Mendukung berbagai format (JSON, Parquet, Avro)	Juga mendukung banyak format, tetapi melalui MapReduce
Pemrosesan Streaming	Mendukung pemrosesan streaming real-time dengan Spark Streaming	Hadoop Stream dapat digunakan, tetapi tidak seefisien Spark
Dukungan untuk Machine Learning	Memiliki MLlib untuk machine learning yang terintegrasi	Terpisah, menggunakan alat seperti Mahout atau TensorFlow
Arsitektur	Menggunakan Directed Acyclic Graph (DAG) untuk eksekusi	Berbasis MapReduce dengan fase map dan reduce
Disposisi Sumber Daya	Mendukung dinamisitas melalui cluster manager seperti YARN, Mesos, atau Kubernetes	Memanfaatkan YARN sebagai manajer sumber daya

Penggunaan Sumber Daya	Meminimalkan penggunaan disk dengan komputasi di memori	Sering kali memerlukan ruang disk lebih banyak
Ketahanan	Kinerja mungkin lebih baik dalam ketersediaan tinggi, terutama di cloud	Dikenal dengan replicating data di HDFS untuk ketahanan

---

Spark Executor

1. Menerima Task:
    - Menerima task dari driver sesuai dengan rencana eksekusi yang ditentukan dalam aplikasi Spark.
2. Menjalankan Task:
    - Melaksanakan task yang diberikan dengan memproses data dari partisi yang relevan.
3. Mengelola Memori:
    - Mengelola penggunaan memori untuk menyimpan data sementara, termasuk hasil intermediate dari task yang sedang dieksekusi.
4. Mengembalikan Hasil:
    - Mengembalikan hasil pemrosesan task ke driver setelah eksekusi selesai.
5. Mengelola Kesalahan:
    - Mengatasi kesalahan yang terjadi selama eksekusi task dan melaporkan kembali ke driver.
6. Berinteraksi dengan Storage:
    - Mengakses sistem penyimpanan seperti HDFS atau sistem penyimpanan lainnya untuk membaca dan menulis data sesuai dengan kebutuhan task.
7. Menjalankan Multi-threading:
    - Menjalankan beberapa task secara paralel di dalam satu executor (tergantung pada konfigurasi dan sumber daya yang tersedia).
8. Menangani Shuffle:
    - Mengatur data yang diperlukan untuk shuffle operation, yang termasuk memindahkan data antar executor jika diperlukan.
