11: Q88 – Q91 Read-Write Vs Append-Only File Systems

Posted on January 18, 2017

Q88. How will you modify a portion of an HDFS file?
A88. HDFS is an "**append-only**" file system. The most common use case of Hadoop data ingestion is to append new sets of event-based and/or sub-transactional data. The large data processing applications are typically built around the premise that things don't change. Hence to modify any portion of a file, you must rewrite the entire and replace the old file. This is true even if you want to modify (i.e. update or delete) a single byte.

For example, if you are building a HDFS based Hive table to store transactional data, new purchases can be added to the table. This works fine in an "append-only" file system. When someone cancels an order or want to adjust the quantity on the order, instead of updating the existing record, create a new adjustment or delta record to indicate the modification.

```
1
2  CREATE EXTERNAL TABLE Purchases {
3    txn_id BigInt,
4    order_id BigInt,
5    product_id BigInt,
6    qty Int,
7    datetime timstamp
8  }
9
10
```

When you update an order, you can create a new record with a new "txn_id" and the same order_id. You can also bump the datetime. This will record a series of transactions for a given order.

Q89. How does HBase handle modifications like editing or deleting a record?
A89. When writing records in HBase, these records are going to land in a file in the HDFS. When you add records as shown below, it will have minimal impact to the system as HDFS is an append-only file system.

300+ Java Interview FAQs

300+ Java FAQs 🔥
16+ Java Key Areas Q&As
150+ Java Architect FAQs
80+ Java Code Quality Q&As
150+ Java Coding Q&As

300+ Big Data Interview FAQs

300+ Big Data FAQs 🔥

FAQs Data - 01: SQL
FAQs Data - 02: Data Modelling
FAQs Data - 02: Data Warehouse
FAQs Data - 03: Big Data
FAQs Data - 04: Hadoop (HDFS)
FAQs Data - 05: MapReduce
FAQs Data - 06: Hive
FAQs Data - 07: Impala
FAQs Data - 08: Spark
FAQs Data - 09: Spark SQL
FAQs Data - 10: Apache Kafka
FAQs Data - 11: Data Governance
FAQs Data - 11: NoSQL
FAQs Data - 12: Data security
FAQs Data - 13: Analytics & Science
FAQs Data - 14: AWS
FAQs Data - 15: Sqoop & Nifi
FAQs Data - 16: Yarn, Zookeeper
FAQs Data - 40: Scala 140+ FAQs

HBase Records:

```
Record 1 = John
Record 2 = Peter
Record 3 = Sam
```

What if you want to edit the "Record 2"? HBase has a concept named "**tombstone**". This concept is there to say that "Record 2" in the HFile is no longer valid, and if the versioning on the column family is set to greater than 1, a new tombstone entry will be created within the same file.

Tombstone records:

```
Record 2 = Liam
```

Deletion of a record also creates a tombstone entry.

Q90. What if the tombstone records grow really large?
A90. The **compaction process** that runs periodically or manually reads the tombstoned records, and yields a **new HBase file**. A new HBase file is created as HDFS is an "append-only" system.

The compaction process can adversely impact latency as the HBase tables need to be locked whilst a new file is being created from an old file, and then the new file needs to be swapped over for the old file.

Q91. What are the different approaches to perform updates/deletes in in the BigData space?
A91.

#1. Spark job to Merge, Compact and Update

Write an Apache Spark job to

1) Load the initial bulk data. This is the **master data** (aka snapshot).

2) Load the newly updated data. This is the **delta data** containing updates & deletes.

3) Merge the master and delta data by the join field. For example, in Spark read master & delta as 2 separate dataframes, and then "union" both the dataframes and apply the "windowing" function which "partition" by the join field and order by descending timestamp to rank() the records. Pick the rank 1 for each join field.

4) Write the merged data to a temporary folder if you are not able to over-write the existing directory.

FAQs Data - 41: 100+ Python FAQs
Tutorials - Big Data

800+ Java Interview Q&As
300+ Core Java Q&As
300+ Enterprise Java Q&As
150+ Java Frameworks Q&As
120+ Companion Tech Q&As
Tutorials - Enterprise Java

5) Over-write the master data folder with the temporary data folder.

More complex to implement compared to the other approaches. The processing SLAs depend on the cluster size.

#2. NoSQL store like HBase

HBase provides native support for updates. You can ingest data into HBase via ETL tools and real-time streaming. The data on HBase can be queried via SQL by providing a Hive table on top of it.

```
1
2  CREATE EXTERNAL TABLE log_entries (
3  id string
4  ,log_payload string
5  ,log_tmstmp timestamp
6  ,log_lvl string
7  )
8  STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
9  WITH SERDEPROPERTIES ('hbase.columns.mapping' =
10 ':key
11 ,i:log_pl
12 ,i:log_tmstmp
13 ,i:log_lvl
14 )
15 TBLPROPERTIES ('hbase.table.name' = 'tbl_log_entries');
16
```

Good for any size data with low latency processing requirements. Good for simple updates.

#3. A hybrid approach with RDBMs & HDFS

This approach uses RDBMS & ETL processing to maintain editable data and then periodically move that data to HDFS via Sqoop to replace previous versions of the data.

Downside of this approach is performance degradation when you need to ETL billions of records from RDBMs into HDFS.

This approach is more suited for small to medium size data with high processing latency requirements.

#4. “Subset merge, compact & update

If your data volume is so large that any of the above approaches cannot complete within the given SLA (Service Level Agreement) window without investing in more nodes, you can apply this “Good Enough” strategy, which tries to partition data based on the likelihood of getting updated. For example, partitioning **subset** of the data based on record creation or last modified timestamp meaning that recent records are more likely to be updated.

So, instead of applying the “#1 Merge, Compact & Update” approach to the entire dataset, you are applying only to subset to improve performance. This

approach should be used only when the full merge, compact, and update strategy cannot be used.

More suited for extremely large volume of data.

‹ 10: Q80 – Q87 HBase Schema Design Interview Q&As

07: spark-xml to split & read very large XML files ›

Arulkumaran

Mechanical Engineer to self-taught Java engineer within 2 years & a **freelancer** within 3 years. Freelancing since 2003. Preparation empowered me to **attend 190+ job interviews** & choose from **150+ job offers** with sought-after contract rates. Author of the book "**Java/J2EE job interview companion**", which sold **35K+ copies** & superseded by this site with **2,050+** registered users. Amazon.com profile | **Reviews** | LinkedIn | LinkedIn Group | YouTube

Contact us: java-interview@hotmail.com

Disclaimer