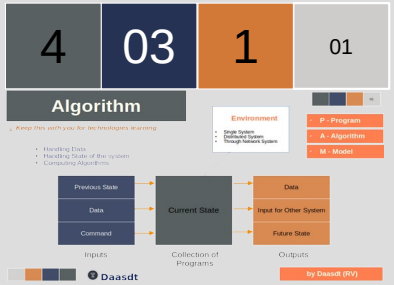
DS Avro format



- Encode data in Binary
- Text based metadata alongside of binary
  - But Binary itself Serialized
- Schema Based & Stored in JSON
- Schema Evolution Support
- Serialized format (convert to byte stream)
- Distributed Storage and Usage

- Binary Serialization Format

- Write Optimized

Daasdt

by Daasdt (RV)

DS Avro format

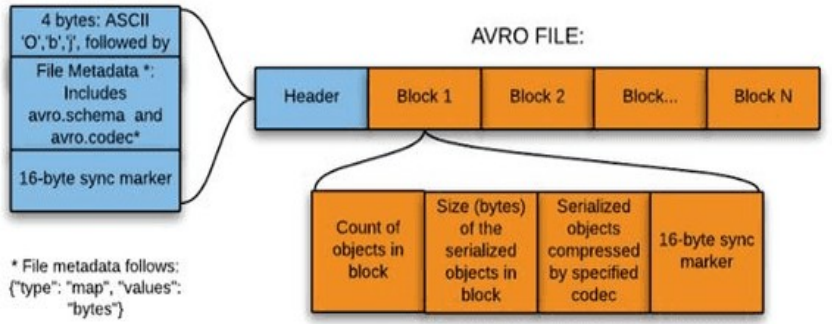
Environment
Distributed Environment

Data Input Output ↔ **Avro Algorithm** ↔ **Stored in Binary Serialization**

- .avro
- Standalone
- Distributed – HDFS &S3
- Message Q - Kafka

- Schema Layer : Define Schema and Data type
- Serialization / De Serialization Layer : Conversion
- Data Layer : Binary Records organized in blocks
- Integration Layer : Integrate with big data tools

How Avro Stores Data



Header: Schema and metadata.

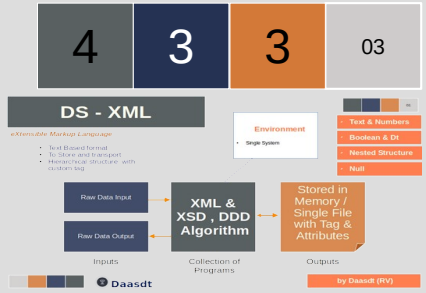
Data Block 1: Serialized data + sync marker.

Data Block 2: Serialized data + sync marker.

Daasdt

by Daasdt (RV)

DS Avro format



- Compact & Efficient
- Schema Evalution
- Compression
- Optimized of distributed
- Write Heavy Workloads

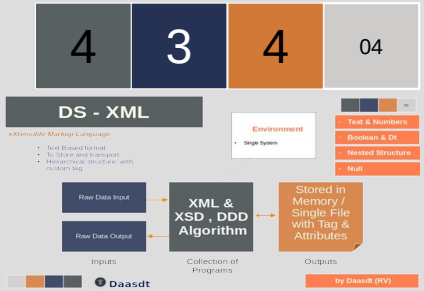
Python:
avro ,fastavro

Go:
go get github.com/linkedin/goavro/v2

Daasdt

by Daasdt (RV)

DS Avro format

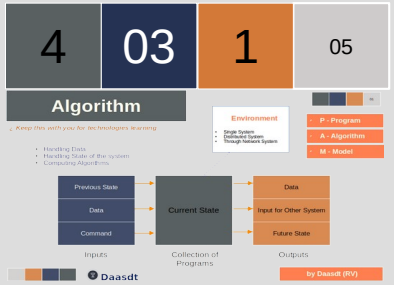


Coding ...

Daasdt

by Daasdt (RV)

DS ORC format



- Optimized Row Columnar
- Using Advance compression technique
- Column wise group compression
- Supports lightweight index (min, max,sum)
  - Predicate push down (Skip row group)
- Split across multiple nodes
- Schema evaluation and metadata
- Run-length encode / Dictionary encode

- Binary Serialization Columnar Format

- Distributed parallel processing

Daasdt

by Daasdt (RV)

DS ORC format

Environment
Distributed Environment

- **ORC limit :**
- **Write heavy**
- **Complex update**
- **In Memory**

Data Input Output ↔ **ORC Algorithm** ↔ **Column Stored in Binary Serialization**

- Serialized Layer : Conversion into Byte Stream
- Strips Layer : Data split into column and stored in strip
- Compression : Compression , Encoding & Indexing
- Metadata: add in footer and post scripts

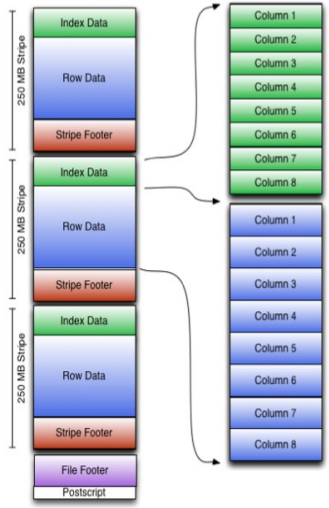
Header: Identity of ORC files

Strips: Primary data storage Unit (64 MB)

- **Index data :** min ,max ,count / Column
- **Raw data :** Column data in compressed binary format
- **Stripe Footer** :Location and type of information for each column

Footer : eof - Strips offset , column Type . Contains metadata for file.

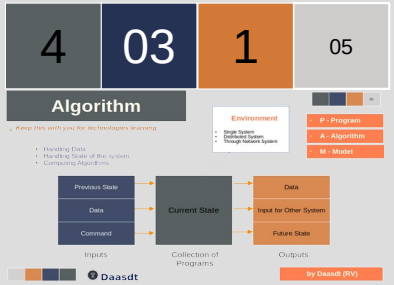
Post scripts : Last segment of file Compression algorithm , file length



Daasdt

by Daasdt (RV)

DS ORC format



- Columnar Storage
  - Each column stored separately (Compression & Pruning)
- Data Encoding & Compression
  - Dictionary Encoding : String & Catogarical data
  - Run Length Encoding : Integer with repeating values
  - Direct Encoding : float and other types
- Indexing & Metadata
  - Column statistics (min, max ,null count , sum)
  - Row Group index – Strips divided into row groups (10,000 rows)- enabled selective reading

- Hive integration
- Spark Integration
- Presto / Trino

Daasdt

by Daasdt (RV)

DS ORC format


```
df = spark.read.orc("output/example.orc")
df.write.orc("output/example.orc", mode="overwrite")
```

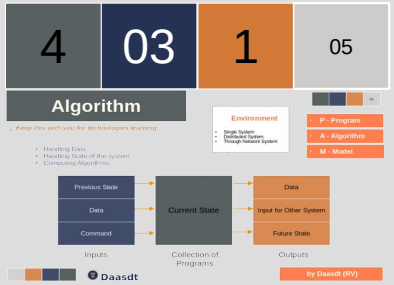
```
data = {
    "name": ["Alice", "Bob", "Charlie"],
    "age": [25, 30, 35],
    "city": ["New York", "San Francisco", "Chicago"]
}
```

```
#high-level API Pandas
df = pandas.DataFrame(data)
df.to_orc("example.orc")
df = pd.read_orc("example.orc")
```

```
#low-level interaction pyarrow
table = pyarrow.table(data)
with pyarrow.orc.ORCFileWriter("example.orc") as writer:
    writer.write(table)
with pyarrow.orc.ORCFile("example.orc") as reader:
    table = reader.read()
    df = table.to_pandas()
```

```
#Distributed processing dask
import dask.dataframe as dd
ddf = dask.dataframe.from_pandas(pandas.DataFrame(data), npartitions=1)
ddf.to_orc("example.orc", engine="pyarrow")
ddf = dask.dataframe.read_orc("example.orc", engine="pyarrow")
```

DS Parquet



- Column wise manner data store
  - Efficient compression & Selective access
- Self descriptive & include schema
- Supports nested structure : Array & Map
- Read only column needed
- Schema evaluation and metadata
- Column Compression - Snappy default
- Predicate push down using stats

- Binary Serialization Columnar Format
- Distributed parallel processing

Daasdt

by Daasdt (RV)

DS Parquet

Environment
Distributed Environment

Data Input Output ↔ **Parquet Algorithm** ↔ **Column Stored in Binary Serialization**

- **Parquet limit :**
- **Write heavy**
- **Smaller file**
- **In Memory**

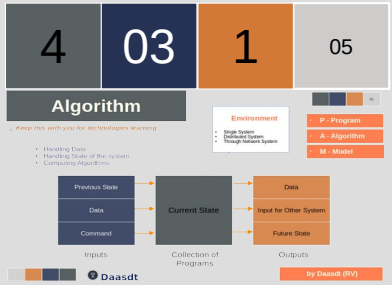
- Serialized Layer : Conversion into Byte Stream
- Row Group : Rows split into row groups
- Column Chunk : Each column compressed and divided into pages
- Footer & Metadata: Metadata added into footer
- File Construction: Header , row groups , footer & Magic numbers are written into desk

- Footer Read : Metadata & Schema load from footer
- Selective column access : Column pruning
- Row group filtering : Predicate push down
- Page decompression : Relevant pages decompressed

Daasdt

by Daasdt (RV)

DS Parquet File Format



High Level Layout

[File Header][Row Groups][File Footer][Magic Number]

- Small Indicator
- Footer location for metadata

- Main data blocks , each contain multiple rows
- Each row group processed parallel

Metadata

Unique id at start and end of file

Row Group Structure

[Column Chunk 1][Column Chunk 2]... [Column Chunk N]

- Each column of row group is stored as continuous block

Column chunk Structure

[Page 1][Page 2]... [Page N]

- Stores data for a single column within a row group
- Organized as pages(8 to 32 kb)
  - Data pages :Actual data
  - Dictionary Pages
  - Index Pages

Daasdt

by Daasdt (RV)

DS Parquet