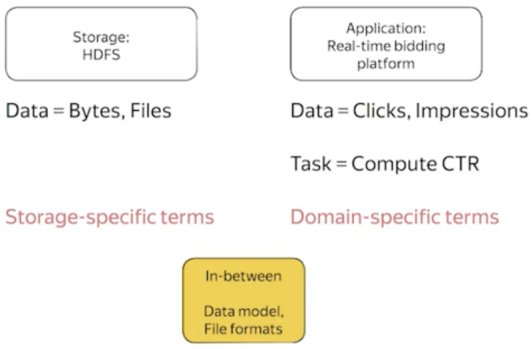
Week1: Tuning Distributed Storage Platform with File Types @Sieun Bae

Data modeling and file formats

- data storage
- trailor



RTB example

- CTR: Click Through Rate

데이터 처리 방식을 선택하는 것은 어플리케이션의 정확성, 성능, 유연성, 유지 관리 능력에 영향을 미침

Data modeling

data model

a way you think about your data elements, what they are, what domain they come from, how different elements relate to each other, what they are composed of

- abstract model
- explicitly defines the structure of data
- makes some things easier to express than others

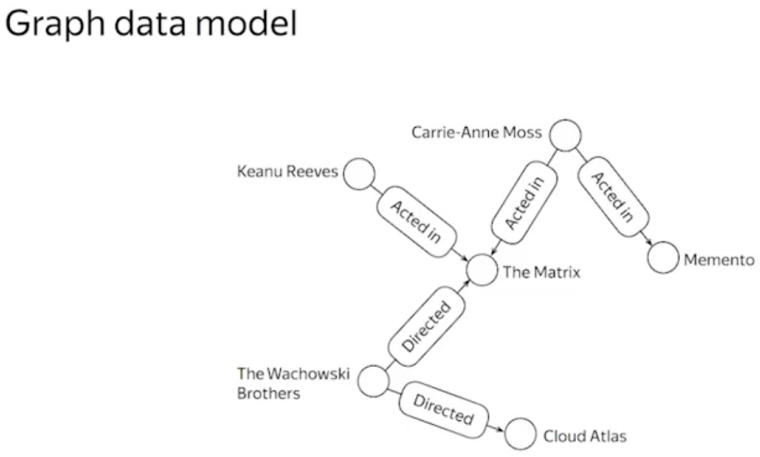
▼ Relational data model

data set (table, relation), tuples (rows), columns (attributes), values

Event	Timestamp	User ID	Ad ID
IMPRESSION	T21:04:13	u1248	a864
IMPRESSION	T21:04:15	u3192	a711
CLICK	T21:04:20	u3192	a711

▼ Graph data model

vertices (entities), edges (relations)



Unstructured data?

- technically, all data is structured at least as a byte sequence
- usually, means "not structured enough for a task"

비교적 복잡성이 클 뿐 정형, 비정형은 사실 없다.

› Ex. 1: Logs = Line per request with all related data
› Easy to work with

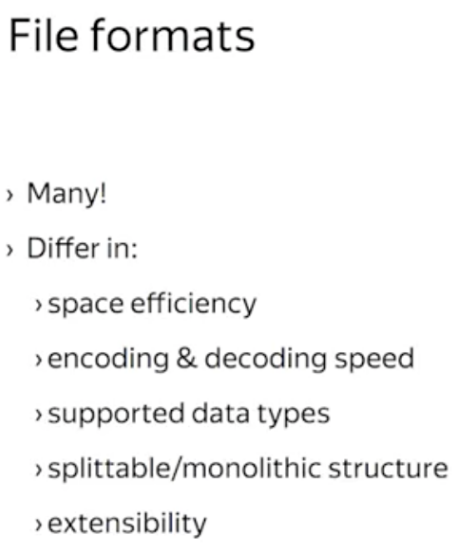
› Ex. 2: Video = Sequence of frames
› Hard to work with

File format (storage format)

- defines (physical) data layout
- different design choices lead to different tradeoffs in complexity

  → affects performance, correctness

- primary function: to transform between raw bytes and programmatical data structures (serialization & deserialization)

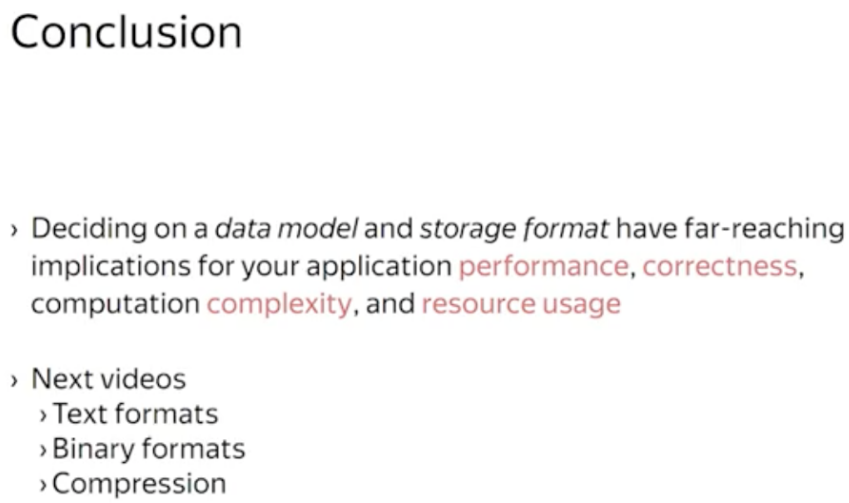


- 다른 형식은 사용되는 디스크 공간에 직접적인 영향을 주는 다른 코딩 체계 사용
- 공간과 시간은 반비례

(공간 절약은 데이터 작업에 필요한 추가 계산 비용을 수반하여 시간을 증가시킴)

ex) Parsing integers 구문 분석

- Cell formats: strict at the right time, force global constraints on data, user's responsiblity to validate data, check constraints
- splittable or monolithic structure: extract subset of data, how to splitt efficiently, compression or encryption..
- extensibility: 호환성 문제



Text formats (csv, tsv, json, xml)

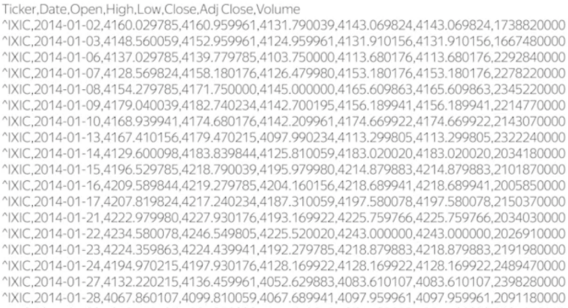
designed for human-readability

human-readable

Example: web server logs

price taken! need to parse it, convert it from the textual form into programmatic data structures

Example: CSV



Comma Seperated Value, Tab Seperated Value

(ex) finances, stock data

- space efficienty: bad
- speed: good
- data types: only strings → 코드부담 높아짐
- splittable: splittable without header
- extensibility: bad

To store floating-point values in CSV format, rather inefficient, due to the consumed space and complex parsing. Parsing floating-point numbers is a non-trivial task, search for "e-308 dtoa bug".

Example: JSON

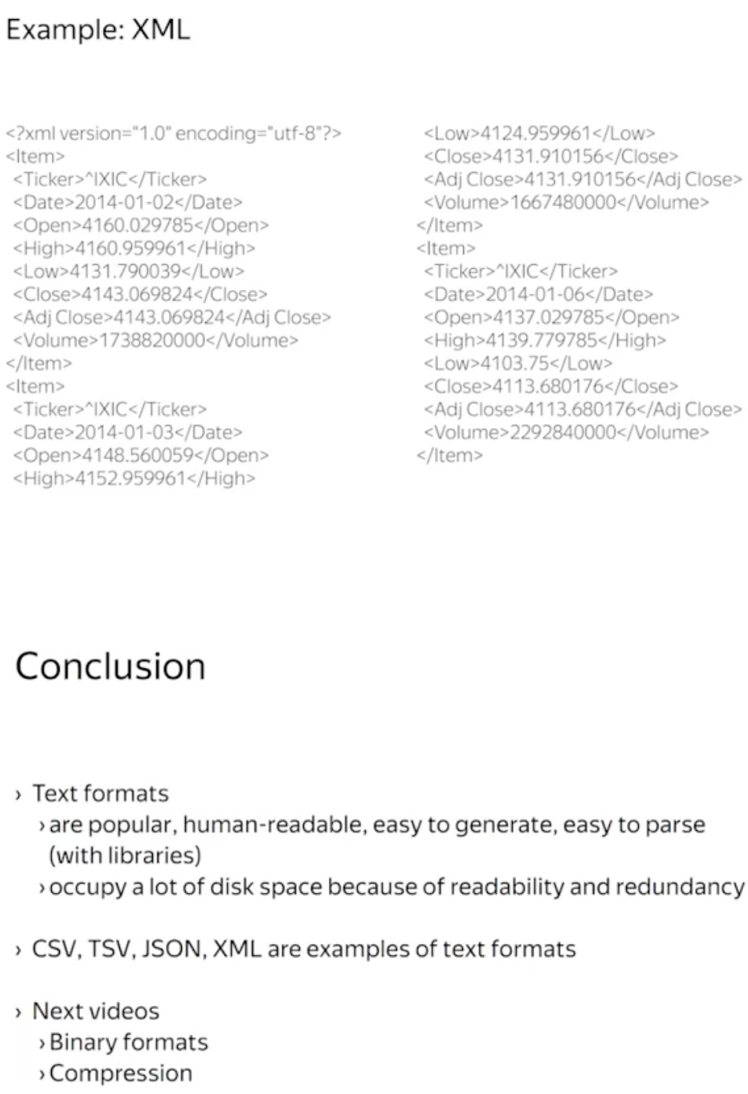
Javascript Object Notation

- 웹에서 많이 사용, 많은 API가 JSON 데이터 생성하고 사용
- python, c++, java..
- space efficiency: bad (worse than csv)
- speed: 가벼운 프로그램에 충분히 좋음 (python: 100-300MB/s, C++ Java: 1GB)
- data types: strings, numbers, booleans, maps, lists, 개발자 입장

에서 편리함

- splittable: if 1 document per line
- extensiblity: yes. 필드 쉽게 추가, 삭제, 필드가 있는지 없는지 유추, 쿼리

in JSON, there is no distinction between floating-point numbers and integral numbers in the specificaiton (go)



Binary formats (sequenceFile, Avro)

designed for machines, record oriented

효율성과 정확성을 가독성과 tradeoff

Inefficiencies of text formats

› To parse "100500"
  › iterate over characters: '1', '0', '0', '5', '0', '0'
  › convert them to digits: 1, 0, 0, 5, 0, 0
  › fold into the result: 1*100000 + 0*10000 + 0*1000 + 5*100 + 0*10 + 0*1

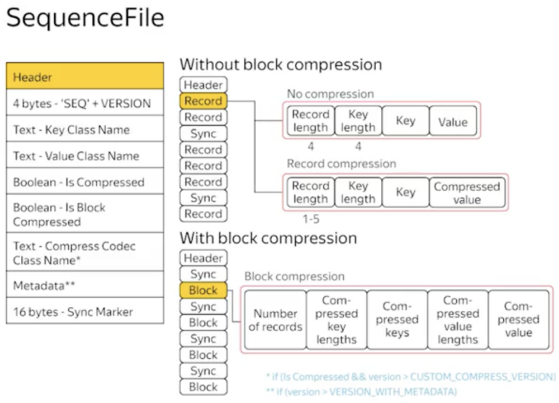
› Not as fast as simple copying

Sequence file

- first binary format implemented in Hadoop
- (ex) 중간 데이터를 MapReduce 계산에 저장
- stores sequence of key-value pairs

  → 다른 언어와의 interoperability를 위한 것이기 보다는 map-reduce 과정에서 그 진가를 알 수 있을 것
- Java-specific serialization/deserialization

  → ad hoc java-specific manner



record compression vs block compression

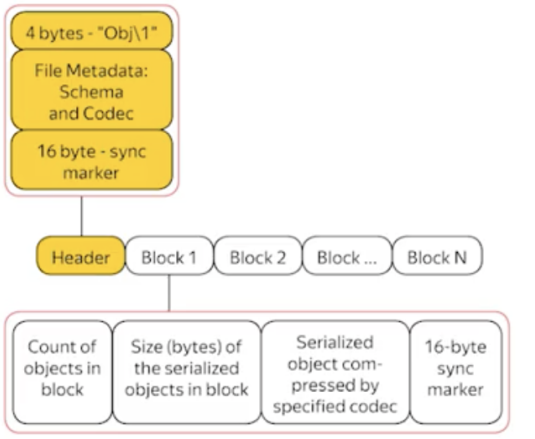
- **record compression**: every value is compressed individually
- **block compression**: a set of keys or values are compressed together resulting in better compression

- space efficiency: moderate. for primitive types, on-disk format closely matches the in-memory format to enable fast encoding and decoding, plug compression
- speed: encoding-decoding speed. good. 기본 값은 그대로 복사
- data types: any types
- splittable: splittable w/ **sync marker**s.

- extensibility: not at all. using version of serializing data or revisions of deserialization code only...

Avro (popular)

- both format & support library
- stores objects defined by the **schema**
  - specifies field names, types, aliases
  - defines serialization/deserialization code
  - allows some schema updates
- design goal: **interoperability** with many languages



serialization code는 user-provided code가 아닌 schema에 의해 구분

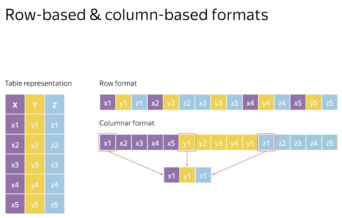
- space efficiency: similar to sequenceFile, encoding-memory format 따라감, compression 통해 공간 save
- speed: good with code generator. generate serialization/deserialization code from a schema.
- data types: same types as JSON, plus a few more (enumerations records)
- splittable: using sync marker
- extensibility: nice. field addition, removal, renaming,

---

analytic workloads, Hive and Impala, **I/O bound**에 걸리는 시간을 줄이기 위해..

- **not reading the data necessary for the processing**
- **using superior compression schemes**

⇒ columnar data formats

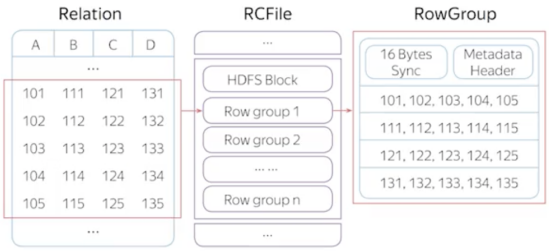


- relational data model
- **row format**: completely serialize, read values → still need to read the whole table
- **columnar format**: transpost, efficiently scan only the necessary data, achieve better compression ratios (more regular, repetitive), price taken! to reconstruct the full row, need to perform lookups from all the columns which is likely to cause more i/o operations, but by accessing this subset of col, reduce

RCFile (Record Columnar)

- most popular storage formats for data warehouses
- first columnar formats in Hadoop
- 행이 단일 시스템에 의해 관리되는 단일 블록으로 반복된다고 가정하는 체계에서는 row assembly가 로컬 작업임을 보장 → network access X
- 모든 RCFile은 여러 HDFS 블록에 걸쳐 있음
- 모든 HDFS 블록 내에 앞에 정의된 행 그룹이 하나 이상 있음

RCFile

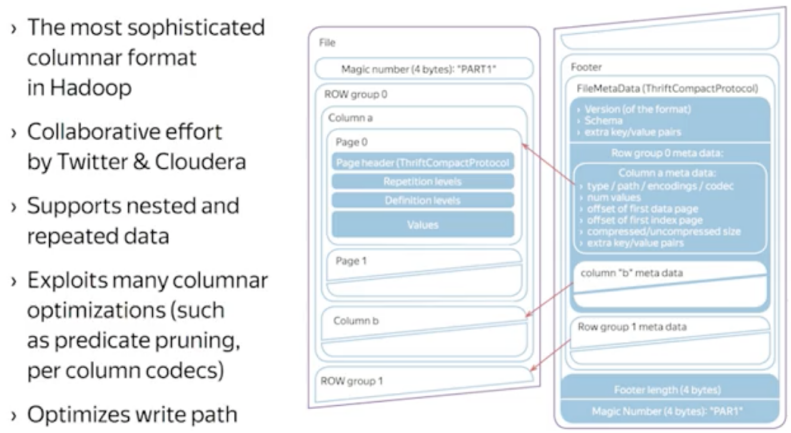


- 모든 행 그룹에는 sync marker, metadata, columnar data가 있음
- horizontal, vertical data partitioning to layout data
  - split rows into row groups
  - transpose values within a row group(columnar)

- **metadata**
  - # of rows, # of cols, total # of bytes, bytes per col, bytes per value → used by decoder to read the consequent col data.
  - compressed w/ the run-length encoding → save on the repeated integers.

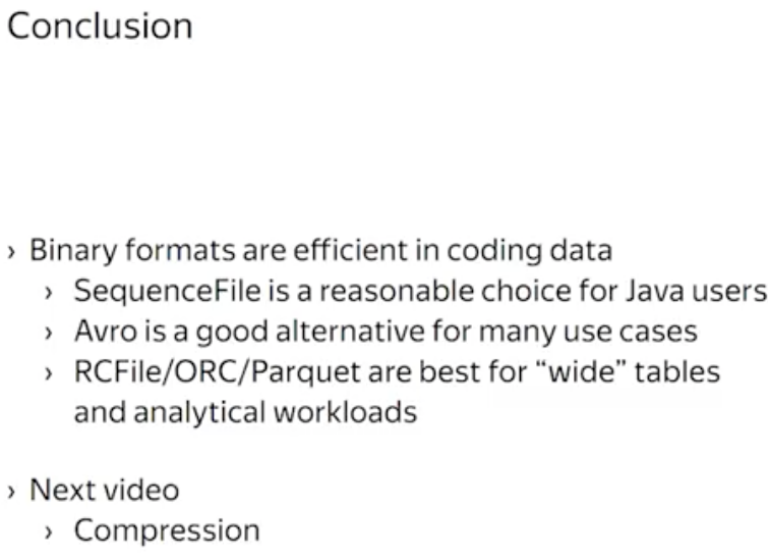
    - col is compressed w/ a general-purpose codec (ZIP)
- to produce a block of data, need to buffer a row group within the main nemory and transpose it, and then precompute metadata.
- space efficiency: good (columnar), 데이터 자체 블록단위 압축
- speed: moderate, less IO를 통해 속도 빠르게
- data types: byte strings. untyped. (serialization, deserialization을 처리하는 Hive와 함께 주로 사용하기 때문) one may treat RCFile as a tabular container
- splittable: ok. sync marker
- extensibility: not really. 스스로 스키마 마이그래이션을 다루어야함 (Hive rewrites data on schema change)

Parquet



key optimization

ex) segment pruning (an ability to skip an entire segment of data)



For general-purpose **data storage and manipulations**

Java users → SequenceFile

Binary alternative to JSON → Avro

For **analytical purposes** like data warehousing → RCFile or Parquet

CF

- csv library for python (Go)
- JSON library for python (Go)
- Apache Avro (Go) Data serialization system
- RCFile paper (Go)
- Apache Parquet (Go)

Compression

binary > string: transcoding speed and space usage

binary < string: human-friendly, widespread, quick prototyping, but redundancy makes increased disk space usage

- **Block-level compression**
  - used in (Binary formats) SequenceFiles, RCFiles, Parquet
  - applied within a block of data
  - compression does not hinder the ability to naviate through the file quickly ← sync marker, metadata in RCFile
- **File-level compression**
  - 기록되기 전 파일이 압축된 것 (zip)
  - applied to the file as a whole
  - hinders an ability to navigate through file
  - 일반적이므로 MapReduce를 포함한 경우 파일에 대한 사소한 분할 없이도 framework 자체적으로 decompress 가능

Bzip2 slowest yet the most efficient

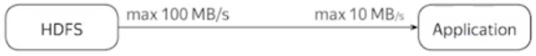
Snappy fastest, yet not so efficient

Codecs

- Gzip
  - compression speed ~16-90 MiB/s
  - decompression speed ~250-320 MiB/s
  - ratio ~2.77 .. 3.43
- Bzip2
  - compression speed ~12-14 MB/s
  - decompression speed ~38-42 MiB/s
  - ratio ~4.02 .. 4.80
- LZO
  - compression speed ~77-150 MiB/s
  - decompression speed ~290-314 MiB/s
  - ratio ~2.10 .. 2.48
- Snappy
  - compression speed ~200 MiB/s
  - decompression speed ~475 MiB/s
  - ratio ~2.05

dataset에 따라 다르긴 함

When to use compression?



if process data at a rate 10MB/s

HDFS에서 빠르게 제공해도 App에서 처리하는 속도가 느리기 때문에 효과적이지 않음

No benefit in using compression

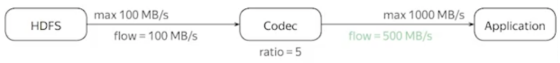
CPU-bound!

spends more time computing rather than doing IO

Adding compression would put more pressure on CPU and increase the completion time

IO-bound!

more time waiting for IO, rather than doing actual computation



Five times more throughput when using compression

Adding compression would allow HDFS to stream the compressed data at rate 100, which transforms to 500 of uncompressed data, assuming the compression ratio of five → 5배 빠른 계산

Conclusion

- Raise awareness about application bottlenecks
  - CPU-bound → cannot benefit from the compression
  - I/O-bound → can benefit from the compression
- Codec performance vary depending on data, many options available

Conclusion (lesson)

- Many applications assume relational data model
- File format defines encoding of your data
  - text formats are readable, allow quick prototyping, but inefficient
  - binary formats are efficient, but more complicated to use
- File formats vary in terms of space efficiency, encoding & decoding speed, support for data types, extensibility
- When I/O bound, can benefit from compression
- When CPU bound, compression may increase completion time

Big Data and Distributed File Systems (Quiz)