Факултет техничких наука

Универзитет у Новом Саду

Архитектуре система великих скупова података

Серијализациони и компресиони формати у системима великих скупова података

Аутор:
Маја Благић

Индекс:
Е2 83/2022

25. април 2023.

Сажетак

Серијализациони и компресиони формати су кључни елементи у системима великих скупова података (енгл. *Big Data*), који помажу у ефикасном смјештању, преносу и обради података. Овај рад ће анализирати и поредити различите особине поменутих формата и дати приједлоге када их треба користити.

Садржај

1 Увод **1**

2 Hadoop **2**

3 Серијализација **3**
3.1 Thrift i Protocol Buffers 3
3.2 Apache Avro 5

4 Компресија **7**
4.1 Примјер компресије 8

5 Закључак **10**

Списак изворних кодова

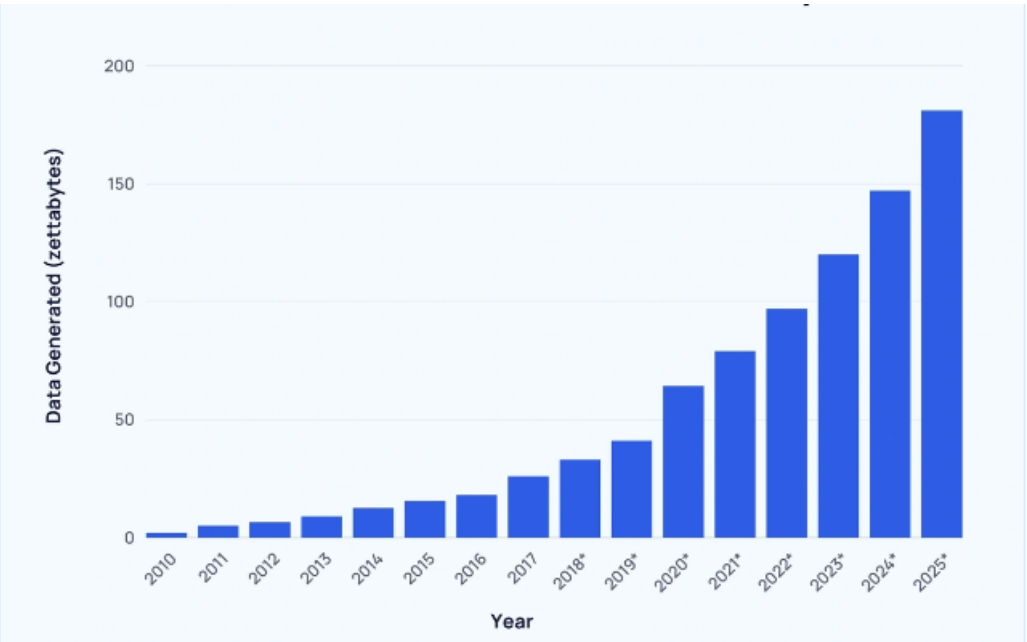
Списак слика

1 Количина података генерисана од 2010. до 2025. године 1
2 Шема дефинисана помоћу Трифт формата 4
3 Шема дефинисана помоћу Протокол Бафер формата 4
4 Енкодирана бајт секвенца настала користећи Трифт формат 5
5 Авро *IDL* шема 6
6 Енкодирана бајт секвенца настала користећи Авро формат 6
7 Компресиони формати у Хадупу 7
8 Јава класа која користи gzip компресију 9
9 Скрипта за компресију фајла 10

Списак табела

1 Увод

Појава интернета је донијела са собом појаву велике количине података. Количина генерисаних података расте из године у годину али се процјењује да је 90% података у свијету генерисано у само двије посљедње године. У односу на 2023. годину када је количина генерисанох података била 120 зетабајта, 2025. година очекује пораст до 181 зетабјата што чини повећање за 150% [1].



Слика 1: Количина података генерисана од 2010. до 2025. године

Као посљедица, системи великих скупова података (енг. *Big Data*) постали су неизоставан део скоро сваке организације која обрађује велике количине података. Овај приступ захтева ефикасно смештање и обраду података, како би се остварила брза и ефикасна анализа.

Серијализациони и компресиони формати су кључни елементи у системи великих скупова података, који помажу у ефикасном смештању, преносу и обради података. Серијализација је процес претварања података у низ бајтова који може да се преноси и складишти, док је компресија процес смањивања величине података. Серијализација помаже у преносу података између различитих система, док компресија омогућава ефикасније складиштење података на диску.

Постоје различити формати за серијализацију и компресију података, као што су авро (енг. *Avro*), паркет (енг. *Parquet*), гзип (енг. *gzip*) и снепи (енг. *snappy*). Одабир одговарајућег формата зависи од конкретних потреба и услова у окружењу у коме

се подаци обрађују. Идућа поглавља ће анализирати и поредити различите особине поменутих формата и дати приједлоге када их треба користити.

2 Hadoop

Довољно је завирити у телефон и погледати количину фотографија које складиштимо свакодневно. Затим, сасвим је логично запитати се какво складиште података се може искористити да би се сачувале све фотографије усликане током године али и деценије?

Осим проблема складиштења јавља се порблем читања података. Иако су капацитети дискова за складиштење података (енгл. *Hard Drive*) значајно порасли током година, брзина приступа - односно, брзина којом се подаци могу читати са дискова није драстично порасла. Типичан диск из 1990. године могао је смјестити 1.370 мегабајта (енгл. MB) података и имао је брзину преноса од 4,4 мегабајта по секунди (енгл. MB/s), па бисте могли прочитати све податке са пуног диска за око пет минута. Преко 20 година касније, терабајтни дискови су уобичајени, али брзина преноса је око 100 MB/s, те треба више од два сата да се прочитају сви подаци са диска. Очигледан начин да се смањи време је да се чита са више дискова одједном. Замислите да имамо 100 дискова, од којих сваки садржи сто теорабајта података. Радећи паралелно, подаци би се могли прочитати за мање од две минуте. Постоје и други изазови који се јављају приликом читања и писања података паралелно с више дискова. Први изазов који треба ријешити су хардверски кварови. Чест начин избјегавања губитка података је репликација, односно чување резервних копија података у систему тако да постоји доступна друга копија у случају отказа. Други проблем је да већина задатака анализе захтјева спајање података на неки начин, и подаци прочитани с једног диска могу захтјевати спајање с подацима с било ког од осталих 99 дискова.

За све поменуте проблеме Хадуп (енгл. *Hadoop*) нуди рјешење. Хадуп је радно окружење (енг. *Framework*) отвореног кода који служи за обраду и анализу великих скупова података (енг. *Big Data*). Његова основна функција је обрада података који су распоређени на више рачунара, односно кластера, и то на начин који је паралелан и расподјељен. Хадуп се састоји из два основна модула: Хадуп фајл систем (енг. *HDFS*) за складиштење података и Хадуп *Map Reduce* модела програмирања за обраду података. Он је погодан за рад са великим количинама података, као и за обраду и анализу података који су различитих формата и типова. Сви примјери серилизационих и компресионих формата ће бити извршени и приказани на Хадуп платформи [3].

3 Серијализација

Кад је узмемо у обзир писање и читање података у дистрибуираном окружењу, веома је важно обратити пажњу на ефикасност преноса тих података. Серијализација је процес конвертовања структурираних објеката у низ бајтова, ради преноса тих података преко мреже или ради уписивања у постојеће складиште. Десеријализација је процес којим се обрнуто, из низа бајтова, добијају структурирани подаци. У дистрибуираним системима, серијализација се користи у две различите области: за комуникацију између процеса и за перзистентно складиштење података. У Хадуп радном окружењу, комуникација између чворова је имплементирана коришћењем удаљених процедуралних позива (енг. *RPCs*). *RPC* протокол користи серијализацију да би превео поруку у бинарни низ који се шаље удаљеном чвору. Удаљени чвор потом десеријализује бинарни низ и добија почетну поруку. За *RPC* серијализацију, пожељено је да формат буде:

- Компактан: Компактан формат најбоље користи мрежни пропусни опсег (енг. *bandwidth*), који је дефицитаран ресурс у дата центру.
- Брз: Комуникација између процеса је кључна за рад дистрибуираних система, стога је битно да серијализација и десеријализација имају што краће вријеме трајања.
- Проширљив: Протоколи се мијењају током времена да би задовољили нове захтјеве, стога је битно да су лако прошириви.
- Интероперабилност: Понекад је потребно подржати више сервера написаних на различитим програмским језицима.

Хадуп користи свој формат серијализације, *Writables*, који је свакако компактан и брз, али није тако једноставан за проширивање или коришћење за различите програмске језике осим Јаве. Међутим, постоје и други формати који се могу користити у Хадупу, укључујући Апаче Авро (енг. *Apache Avro*), Трифт (енг. *Thrift*) и Протокол Баферс (енг. *Protocol Buffers*), који су дизајнирани да буду ефикаснији и флексибилнији у односу на *Writables*. Ови формати омогућавају корисницима да дефинишу своје типове података и генеришу код за серијализацију и десеријализацију у више програмских језика [3].

3.1 Thrift i Protocol Buffers

Апаче Трифт (енг. *Apache Thrift*) и Протокол Бафер-и (енг. *Protocol Buffers*) су библиотеке за бинарно кодирање засноване на истом принципу. Протокол Бафери су првобитно развијени у *Google*-у, а Трифт у *Facebook*-у, а оба су постала отвореног кода 2007-08. године.

Трифт као и Протокол Бафер-и захтјевају дефиницију шеме по којој ће подаци бити енкодирани. Оба протокола посједују алате за генерисање кода на основу дефинисане шеме. Алати генеришу класе које могу бити имплементиране на више различитих програмских језика. На сљедећој слици приказана је дефиниција Трифт шеме.

```
struct Person {
  1: required string       userName,
  2: optional i64          favoriteNumber,
  3: optional list<string> interests
}
```

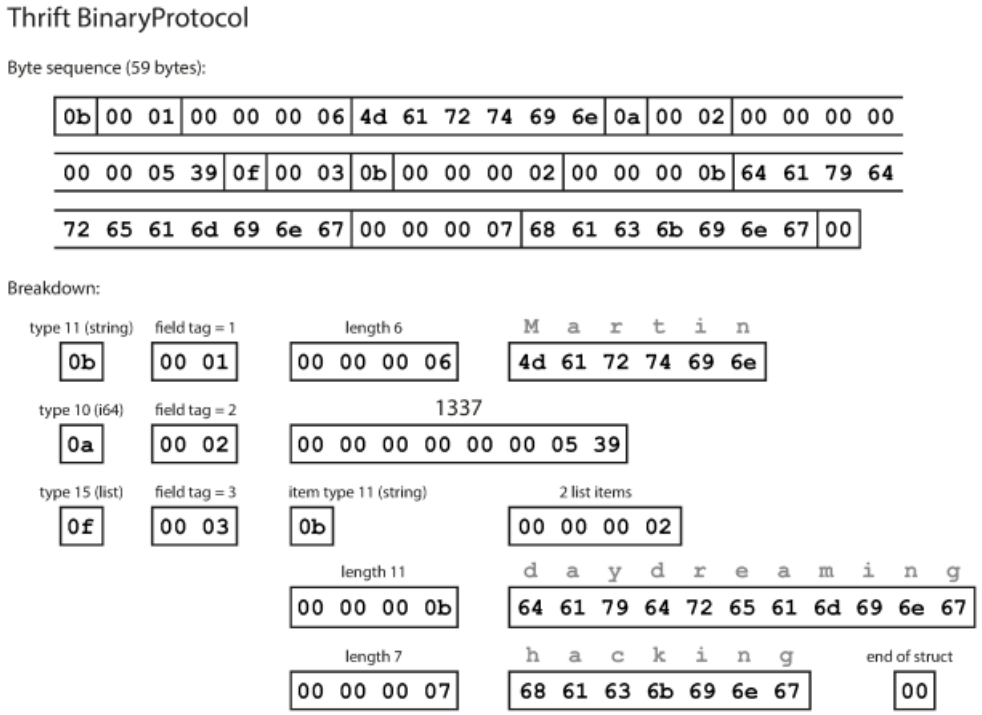
Слика 2: Шема дефинисана помоћу Трифт формата

Еквивалентна шема дефинисана користећи Протокол Бафер приказана је на слици 3.

```
message Person {
    required string user_name       = 1;
    optional int64  favorite_number = 2;
    repeated string interests       = 3;
}
```

Слика 3: Шема дефинисана помоћу Протокол Бафер формата

Када се примјер са слике 2 енкодира, добија се секвенца бајтова која је приказана на сљедећој слици.



Слика 4: Енкодирана бајт секвенца настала користећи Трифт формат

Битно је уочити да свако поље има ознаку типа, и, где је потребно, ознаку дужине. Низови знакова који се појављују у подацима нпр. "Martin", "daydreaming", "hacking" кодирани су као аски (енг. *ASCII*) знакови. Оно што је битно уочити јесте да нема имена поља нпр. *userName*, *favoriteNumber*, *interests*. Умјесто тога, енкодирани подаци садрже ознаке поља, као што су бројеви 1, 2 и 3. То су бројеви који се појављују у дефиницији шеме. Ознаке поља су као алијаси за поља - компактан начин да се каже о ком пољу се говори, без потребе да се наводи име поља [2].

3.2 Apache Avro

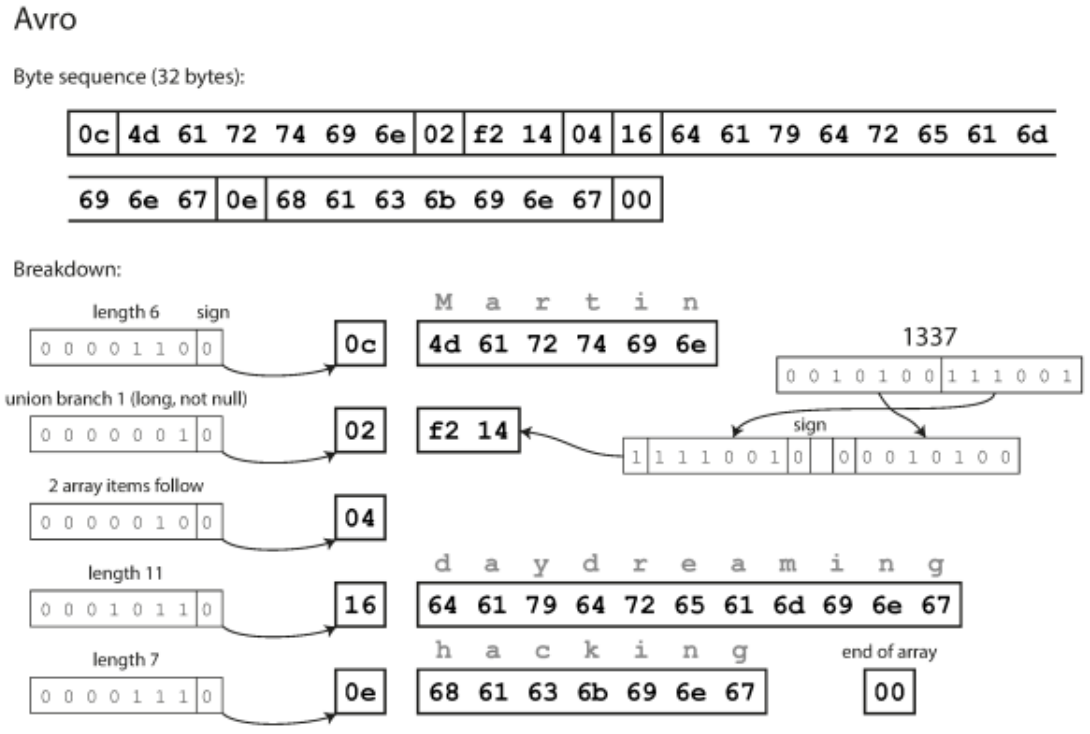
Апаче Авро је систем серијализације података који је независан од програмског језика. Пројекат је створио Даг Катинг (творац Хадупа) да би се суочио са главним недостатком Хадупових "Writables" - недостатком преносивости програмског језика [3].

Авро има два језика шема: *Avro IDL* који је предвиђен за уређивање од стране људи, као и још један заснован на *JSON*-у који је лакши за читање од стране рачунара. На слици можемо видјети примјер шеме, написан у Авро *IDL*-у.

```
record Person {
    string               userName;
    union { null, long } favoriteNumber = null;
    array<string>        interests;
}
```

Слика 5: Авро *IDL* шема

Када се примјер енкодира, добија се секвенца бајтова која је приказана на сљедећој слици.



Слика 6: Енкодирана бајт секвенца настала користећи Авро формат

На слици се може видјети да у секвенци бајтова нема ништа што би идентификовало поља или њихове типове података. Енкодинг се једноставно састоји од вредности које су конкатениране заједно. Оваква природа Авро формата чини да је његов енкодинг најмањи у поређењу с осталим форматима [2].

Још један значајан механизам Авро формата је шема еволуције. На пример, ново, опционо поље може бити додато у шеми. Нови и стари клијенти ће бити у могућности да читају старе податке, док нови клијенти могу писати нове податке који користе ново поље. Обрнуто, ако стари клијент види ново кодиране податке, он ће игнорисати ново поље и наставити обраду старих података [3].

Једна од главних предности Авро формата је што не користи тагове за означавање типова података, већ имена поља. Ово олакшава процес динамичког генерисања шемњ података, што је од великог значаја у случају да постоји потреба за честим ажурирањем базе података. За разлику, ако би се користили *Thrift* или *Protocol Buffers* за ову сврху, сваки пут када се шема базе података промени, администратор би морао ручно ажурирати мапирање имена колона базе података на ознаке поља [2].

4 Компресија

Компресија датотека доноси двије главне користи: смањује простор потребан за чување датотека и убрзава пренос података преко мреже, на диск или са диска. Када је у питању обрада велике количине података, обе уштеде могу бити значајне, зато је потребно пажљиво размисли о начину коришћења компресије у Хадупу. Постоји много различитих формата, алата и алгоритама за компресију, сваки са различитим карактеристикама. Слика 7 наводи неке од најчешћих који се могу користити у Хадупу.

Compression format	Tool	Algorithm	Filename extension	Splittable?
DEFLATE[a]	N/A	DEFLATE	*.deflate*	No
gzip	*gzip*	DEFLATE	*.gz*	No
bzip2	*bzip2*	bzip2	*.bz2*	Yes
LZO	*lzop*	LZO	*.lzo*	No[b]
LZ4	N/A	LZ4	*.lz4*	No
Snappy	N/A	Snappy	*.snappy*	No

Слика 7: Компресиони формати у Хадупу

Брза компресија и декомпресија обично се постижу науштрб уштеде простора. Алати наведени на слици 7 обично омогућавају кориснику да контролише овај баланс у време компресије. Постоји 9 опција, почевши од -1 која означава оптимизацију за брзину, а -9 оптимизацију за простор. На примјер, следећа команда ствара компресовану датотеку *file.gz* користећи најбржу методу компресије:

```
% gzip -1 file
```

Различити алати имају различите карактеристике компресије. Општи компресиони формат је *gzip*. Он ствара баланс између простора и брзине копресије. *Bzip2* компресује ефикасније од *gzip*-а, али је спорији. Декомпресија *bzip2* формата је бржа од његове компресије, али је и даље спорија од осталих формата. Алати као што су *LZO*, *LZ4* и *Snappy* су око 10 пута бржи од *gzip*-а, али су мање ефикасни у компресији података. Ово значи да, ако се користе ови алати за компресију, добија се већа брзину у процесу компресије и декомпресије података, али ће величиа датотека бити мало већа него приликом коришћења *gzip*-а.

Колона "*Splittable*" на слици 7 означава да ли формат компресије подржава дељење. Формати компресије који подржавају дијељење посебно су погодни за *MapReduce* функције.[3]

4.1 Примјер компресије

У овом поглављу ће бити описана компресија *.csv* фајла користећи *gzip* формат. Слика 8 садржи дефиницију јава класе која учитава *csv* фајл, затим га конвертује помоћу *GZIPOutputStream* објекта и на крају га уписује на жељену путању.

```java
GzipCompress.java > GzipCompress > main(String[])
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.zip.GZIPOutputStream;
import java.io.File;


public class GzipCompress {

    Run | Debug
    public static void main(String[] args) throws IOException {

        String inputFilePath = "./movies.csv";
        String outputFilePath = "./movies.csv.gz";
        File outputFile = new File(outputFilePath);

        // Kreiraj izlazni faj ako ne postoji
        if (!outputFile.exists()) {
            outputFile.createNewFile();
        }
        // Otvori ulazni i izlazni stream
        FileInputStream inputStream = new FileInputStream(inputFilePath);
        FileOutputStream outputStream = new FileOutputStream(outputFilePath);

        // Napravi GZIP izlazni stream
        GZIPOutputStream gzipOutputStream = new GZIPOutputStream(outputStream);

        // Procitaj ulazni fajl i pisi ga u GZIP izlazni stream
        byte[] buffer = new byte[1024];
        int len;
        while ((len = inputStream.read(buffer)) != -1) {
            gzipOutputStream.write(buffer, 0, len);
        }

        // Zatvori stream-ove
        inputStream.close();
        gzipOutputStream.finish();
        gzipOutputStream.close();

        System.out.println("File compressed successfully.");
    }
}
```

Слика 8: Јава класа која користи gzip компресију

На слици 9 приказана је скрипта која садржи све кораке потребне за компресију фајла.

```
> $ compress_run.sh
  # 1. Kompajlirajte a GzipCompress.java fajl pomoću javac komande:
  javac GzipCompress.java &

  # 2. Zatim zapakujte .class fajl u JAR arhivu:
  jar cvf GzipCompress.jar GzipCompress.class &

  # 3. Prebacite na hdfs
  hdfs dfs -put -f movies.csv /user/data &
  hdfs dfs -put -f GzipCompress.jar /user &

  # 4. Pokrenite komandu  kako bi kompresovala movies.csv datoteku
  hadoop jar GzipCompress.jar GzipCompress
```

Слика 9: Скрипта за компресију фајла

У првом кораку се компајлира *GzipCompress.java* фајл помоћу команде:

```
javac GzipCompress.java
```

Затим се добијени .class фајл запакује у JAR архиву:

```
jar cvf GzipCompress.jar GzipCompress.class
```

Након тога потребно је пребацити *jar* фајл на хдфс (енг. *HDFS*) помоћу команде:

```
hdfs dfs -put -f GzipCompress.jar /user
```

На крају слиједи покретање команде за извршавање jar фајла и коначна компресија *movies.csv* датотеке :

```
hadoop jar GzipCompress.jar GzipCompress
```

Иницијална величина фајла од 676,9 MB се смањила на 181,6 MB.

5 Закључак

Сви поменути формати серијализације су дизајнирани за пренос података између различитих система или складиштење података на диск.

Writables је формат који је дизајниран специфично за Хадуп и углавном се користи за интерну комуникацију између различитих чворова у класетру или за складиштење података у Хадуповом фајл систему. Предност овог формата је његова компактност и брзина, али недостатак је што није прошириб и интероперабилан са другим програмским језицима.

Apache Avro је формат који је дизајниран за интероперабилност између различитих програмских језика и платформи. Предност Авро-а је да корисник може дефинисати своје типове података, што омогућава већу флексибилност и проширивост.

Такође, Авро је компактан и брз, а нуди и неке напредне функције као што је шема еволуције.

Thrift је сличан Авро-у, али је фокусиран на перформансе и подршку за велике количине података. Предност *Protocol Buffers*-а је његова ефикасност и брзина, што га чини добрим избором за апликације које захтјевају брзу и поуздану серијализацију и десеријализацију великих скупова података.

Поред формата серијалицације формати компресије допринијети у смањењу величине датотека и убрзавању преноса података. Неки формати су погоднији за Хадуп окружење, као на примјер *Splittable* формати који омогућују паралелизацију обраде података на кластеру. Такође, неки формати попут *LZO*, *LZ4* и *Snappy* су бржи од *gzip*-а, али су мање ефикасни у компресији података.

Коначна одлука о томе који формат компресије и серијализације треба користити зависи од специфичних захтева апликације, као што су величина датотеке, брзина преноса и брзина обраде података. Задатак инжињера је да буде упознат са предностима који сваки формат нуди, те да одлучи који од њих представља најбоље рјешење за проблем.

Библиографија

[1] Fabio Duarte. Amount of data created daily (2023), Apr 2023.

[2] Martin Kleppmann. Designing data-intensive applications.

[3] Tom White. *Hadoop: The definitive guide*. O'Reilly, 2015.