Java binary serialization review

Sirenko A.

December 5, 2014

2014

Serialization

We use serialization in 3 cases:

1. to store;
2. to transmit;
3. to loose;

Main features to consider:

1. CPU usage;
2. serialized size;
3. freedom to change format;
4. laboriousness;
5. platform-dependence;

JDK tools

- Standard serialization: smallest effort / worst performance;
- Externalizable serialization: full control over serialization, define your binary format by hand;

serialization	speed	size	freedom to change	laboriousness	platform dependence
standard	poor	big	average	low	only java
externalizable	fast	small	poor	high	independent

Table : Standard serialization vs Externalizable

Why use something else?

Motivation:

- conveniently simple and fast approaches;
- better adopted for our data: optional params, variable-length numbers, aliases;
- independence from language necessary in some cases.

Here we consider:

- standard serialization;
- externalizable;
- protobuf - `https://code.google.com/p/protobuf/`;
- avro - `http://avro.apache.org/`;
- kryo - `https://github.com/EsotericSoftware/kryo`.

Protobuf

Protobuf info

- needs proto-file with description of your data in primitives;
- java, python, c++ officially supported;
- performant and compact;

.proto file:

```
package tutorial;

option java_package = "com.example.tutorial";
option java_outer_classname = "AddressBookProtos";

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
 }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}

message AddressBook {
  repeated Person person = 1;
}
```

Avro

Avro info

- needs schema-file, can use different schemes to read and write;
- java, python, C
  C++, Ruby;
- schema stored with data ⇒ compact files; no manual field-IDs;
- code-generational optional for static languages;

```
{
    "type": "record",
    "name": "Employee",
    "fields": [
        {"name": "name", "type": "string"},
        {"name": "age", "type": "int"},
        {"name": "emails", "type": {"type": "array", "items": "string"}},
        {"name": "boss", "type": ["Employee","null"]}
    ]
}
```

Kryo

Kryo info

- Used as standard serialization with custom serializer when you need it;
- Only java;
- For custom serializer there are primitives and features (variable-length numbers, positive optimisations);

Automatic serialization $\Rightarrow$ not much difference from standard serialization:

```
Kryo kryo = new Kryo();

Output output = new Output(new FileOutputStream("file.bin"));
SomeClass someObject = ...
kryo.writeObject(output, someObject);
output.close();

Input input = new Input(new FileInputStream("file.bin"));
SomeClass someObject = kryo.readObject(input, SomeClass.class);
input.close();
```

Kryo custom

To customize serializer, submit Serializer to Kryo object or read/write primitives using methods of Input/Output:

```
public class StoredUrl2Serializer extends Serializer<StoredUrl2> {

    @Override
    public void write(Kryo kryo, Output output, StoredUrl2 storedUrl2) {
        output.writeByte(0); // first version of format, placeholder for future versions and optimizations
        output.writeString(this.url.toString());
        output.writeShort(crawlResultCode);
        ...
        output.writeVarInt((int) lastFetchTime, false);
        Kryo.writeObjectOrNull(output, sourceIds, String.class);
        kryo.writeObjectOrNull(output, outLinks2, OutLink[].class);
        ...
        output.writeVarLong(contentLength, false);
    }

    @Override
    public StoredUrl2 read(Kryo kryo, Input input, Class<StoredUrl2> aClass) {
       byte format;
       if ((format = input.readByte()) != 0) {
           throw new RuntimeException("Format not supported: " + format);
       }
       Url url = new Url(input.readString(), false);
       crawlResultCode = input.readShort();
       ...
       long lastFetchTime = input.readVarInt(false);
       @Nullable String sourceIds = kryo.readObjectOrNull(input, String.class);
       @Nullable OutLink[] outLinks = kryo.readObjectOrNull(input, OutLink[].class);
       long contentLength = input.readVarLong(false);
       ...
       return new StoredUrl2(url, ..., contentLength);
    }
```

Standard vs Kryo

Tested on core i5-2450M + SSD
Without compression:

parameter	standard	kryo	change %
read time, msec (1.5m objects)	4999	2395	52% faster
write time, msec (500k objects)	2283	975	67% faster
object size (bytes)	332	264	20% more compact

Table : Standard vs Kryo in planners queue

With snappy:

parameter	standard	kryo	change %
read time, msec (1.5m objects)	5306	2626	51% faster
write time, msec (500k objects)	3406	1250	63% faster
object size (bytes)	110	91	18% more compact

Table : Standard vs Kryo in planners queue

Standard vs. Kryo

- Useless improvement.

Standard vs. Kryo

- Useless improvement.
- Cause we don’t care about speed :)

Standard vs. Kryo

- Useless improvement.
- Cause we don't care about speed :)
- Let's compress more:
  GZip compress well, but too slow. Deflate with *BEST_SPEED* option perform between GZip and Snappy.

```
public KryoStoredUrlWriter create(@NotNull File file)
    throws IOException {

  Deflater def = new Deflater(Deflater.BEST_SPEED);

  return new KryoStoredUrlWriter(
    new DeflaterOutputStream(
      new BufferedOutputStream(new FileOutputStream(file), 4 * 1024),
      def
    )
  );
}
```

Standard vs. Kryo in queue

parameter	standard + snappy	kryo + deflate	change %
read time, msec (1.5m objects)	5292	4150	22% faster
write time, msec (500k objects)	2529	2461	same
object size (bytes)	110	73	33% more compact
reconstruction time planner2v (min)	240	210	12.5% faster

Table : Standard + snappy vs Kryo + deflate

Space economy on planner2v (5 storages + 3 queue backups): 250Gb

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;
  - You want to be unique:)

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;
  - You want to be unique:)
- Use Protobuf if

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;
  - You want to be unique:)
- Use Protobuf if
  - you are working at Google;

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;
  - You want to be unique:)
- Use Protobuf if
  - you are working at Google;
  - Data is processed in different languages;

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;
  - You want to be unique:)
- Use Protobuf if
  - you are working at Google;
  - Data is processed in different languages;
  - Need most performant/compact result;

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;
  - You want to be unique:)
- Use Protobuf if
  - you are working at Google;
  - Data is processed in different languages;
  - Need most performant/compact result;
- Use Externalizable if

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;
  - You want to be unique:)
- Use Protobuf if
  - you are working at Google;
  - Data is processed in different languages;
  - Need most performant/compact result;
- Use Externalizable if
  - you are unique - want implement all the features by yourself;

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;
  - You want to be unique:)
- Use Protobuf if
  - you are working at Google;
  - Data is processed in different languages;
  - Need most performant/compact result;
- Use Externalizable if
  - you are unique - want implement all the features by yourself;
  - already working in Big Bank;

Choice

My subjective opinion to choose serialization:

- Use standard serialization if
  - Someone will kill you if you write code slowly
  - You are making task on job interview in Big Bank
  - Serialization don't affect performance of system
  - You add/remove fields sometimes, but lazy to update serializer.
- Use Kryo if
  - Someone will kill you if you write code slowly;
  - Data is processed by java only;
  - Twice speed of standard serialization is enough;
  - Subzero is your favorite fighter in Mortal Combat.
- Use Avro if
  - Data is processed in different languages;
  - You want to be unique:)
- Use Protobuf if
  - you are working at Google;
  - Data is processed in different languages;
  - Need most performant/compact result;
- Use Externalizable if
  - you are unique - want implement all the features by yourself;
  - already working in Big Bank;
  - watched only first 3 slides of this presentation;