

Serialization

Konstantin Knauf, Solutions Architect

Metrics & Monitoring

Agenda

- Flink's Serialization System
- Custom Serializers
- Debugging Serialization



Flink's Serialization System

Serialization in Flink

- Ingestion Serialization
 - de-/serialization of records read/written from/to external systems (e.g. Kafka)
- Wire Serialization
 - de-/serialization of records exchanged between Flink Tasks
 - same as state serializers
- State Serialization
 - de-/serialization of state objects for storage in RocksDB and checkpoints
 - same as wire serializers



Flink's Serialization System

Supported Types

- Natively Supported Types
 - Primitive Types
 - Tuples, Scala Case Classes
 - POJO Types
 - (AvroTypes)
- Un-Supported Types fall back to Kryo for Serialization



Flink's Serialization System

Benchmark Results for Flink 1.8

Serializer	Ops/s
PojoSerializer	305 / 293*
RowSerializer	475
TupleSerializer	498
Kryo	102 / 67*
Avro (Reflect API)	127
Avro (SpecificRecord API)	297
Protobuf (via Kryo)	376
Apache Thrift (via Kryo)	129 / 112*

* with / without type registration

Disclaimer:

The results are not representative. Serialization performance is highly type-dependent and every user is advised to perform his/her own benchmarks. Regardless of the serializer used simple, non-recursive types are faster than complex, nested types.

```
public static class MyPojo {
    public int id;
    private String name;
    private String[] operationNames;
    private MyOperation[] operations;
    private int otherId1;
    private int otherId2;
    private int otherId3;
    private Object someObject; // used with String
}

MyOperation {
    int id;
    protected String name;
}
```



Custom Serializers



Custom Serializers

Registration with Kryo via ExecutionConfig

- `registerKryoType(Class<?>)`
 - registers type with Kryo for more compact binary format
- `registerTypeWithKryoSerializer(Class<?>, Class<? extends Serializer>)`
 - Provides a default serializer for the given class
 - Provided serializer class must extend `com.esotericsoftware.kryo.Serializer`
- `addDefaultKryoSerializer(Class<?>, Class<? extends Serializer>)`
 - registers a serializer as the default serializer for the given type



Custom Serializers

@TypeInfo Annotation

```
@TypeInfo(MyTupleTypeInfoFactory.class)
public class MyTuple<T0, T1> {
    public T0 myfield0;
    public T1 myfield1;
}

public class MyTupleTypeInfoFactory extends TypeInfoFactory<MyTuple> {

    @Override
    public TypeInformation<MyTuple> createTypeInfo(Type t, Map<String, TypeInformation<?>> genericParameters) {
        return new MyTupleTypeInfo(genericParameters.get("T0"), genericParameters.get("T1"));
    }
}
```



Avro Serializers

- Avro generated classes are automatically serialized using Avro.
- `ExecutionConfig#enableForceAvro()` will serialize all POJOs with Avro using schemas inferred via Avro's Reflect API
- `org.apache.flink:flink-avro:${flink.version}` needs to be on the classpath



Debugging Serialization



Debugging Serialization Issues

- Tune locally in your IDE using a profiler
- `TypeExtractor#createTypeInfo()` as entry point for debugging choice of type serializer
- Useful Methods
 - `ExecutionConfig#disableGenericTypes()`
 - `ExecutionConfig#getDefaultKryoSerializerClasses()`
 - `ExecutionConfig#getRegisteredKryoTypes()`
 - `ExecutionConfig#getRegisteredPojoTypes()`
 - `TypeInformation#of(MyClass.class).createSerializer()`



Object Reuse

Slightly off-topic...

- disabled by default
 - `ExecutionConfig#enableObjectReuse()`
- when enabled
 - streaming: same object is used throughout operator chains
 - batch: more complicated re-use patterns possible
 - <https://ci.apache.org/projects/flink/flink-docs-stable/dev/batch/index.html#object-reuse-enabled>



Exercises



Exercises

Tuning for Throughput

Note: If you have not completed the previous exercise, please check out [ffsf-19-solution-3](#) to proceed.

Exercise 4

- Improve the throughput of `TroubledStreamingJob` by making serialization more efficient.
- Improve the throughput of `TroubledStreamingJob` further by identifying any inefficient user code.





ververica

konstantin@ververica.com

www.ververica.com

[@VervericaData](https://twitter.com/VervericaData)