Table of Contents

[Introduction 2](#_Toc3929226)

[Requirements 2](#_Toc3929227)

[Architecture 2](#_Toc3929228)

[Assumptions 2](#_Toc3929229)

[Flow 2](#_Toc3929230)

[Technologies 2](#_Toc3929231)

[Design 2](#_Toc3929232)

[Environment 3](#_Toc3929233)

[Partitioning 3](#_Toc3929234)

[Extendibility 3](#_Toc3929235)

[Performance 3](#_Toc3929236)

[Templates 3](#_Toc3929237)

[DTO 3](#_Toc3929238)

[Schema 4](#_Toc3929239)

[API 4](#_Toc3929240)

[Folder Hierarchy 4](#_Toc3929241)

[Usage 5](#_Toc3929242)

[Input 5](#_Toc3929243)

[Console Output 5](#_Toc3929244)

[File System Output 6](#_Toc3929245)

# Introduction

The main purpose of this application is to develop a general solution for high performance writing in parallel to multiple parquet files, where partitions are based on a specific time-based interval. The application should be flexible as to support any schema provided by the user.

# Requirements

* Convenient to use from java
* Generic enough so adding support for yet another schema (DTO) is easy
* Microbenchmark

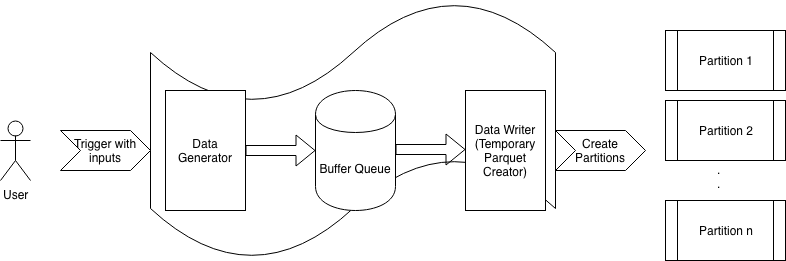
# Architecture

## Assumptions

* Schema must be including timestamp field as the first field

## Flow

This application consists of three main part. Data generation, data persistence and partition creation. As visualized view of the flow as below.



## Technologies

* Apache Spark: Star of the application
* Mojo: Using for maven exec plugin
* Junit: Testing purposes
* Log4j: Logging stuff
* Apache cli: Command line options parser

# Design

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Fill this area

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Producer-Consumer Pattern used for to fill the buffer without blocking data generation and prevent to break the synchronization. Design can be split under three main steps;

* Producer: Data generation part. This part used for generating data according to schema and fill in the buffer for consumer.
* Consumer: Data writer part. This part is used for consuming generated record from the buffer and persist to a temporary parquet writer implementation of API interface. Temporary parquet file used for to use this application with limited resources. Data can be persisted to disk rather than memory to save resources.
* Partitioner: Partitioning part. This part is used for to get temporary parquet file, prepare it according to time intervals provided and generate partitions to the file system. While partitioning Apache Spark is ordering the records. That’s why prevented from repartitioning and preferred to use temporary parquet file as the seeder.

## Environment

* This application designed to run on single JVM.
* Parquet partition files are written to local file system.

## Partitioning

* First column of the parquet file must be named as timestamp and type of long. This field contains data in milliseconds from 1970 with UTC timezone. Name of the partition files are prepared according to this column and must be human readable format.
* Partition intervals must be selected according to timestamp column. This interval shall be configurable as like hourly, per 6 hours etc.

## Extendibility

* Input schema can be provided easily
* Input schema supports the primitive types of the parquet file format
* Input data must be generated according to given schema

## Performance

* API calls should be thread safe
* Buffers, threads and memory usage shall be configurable
* Producer must wait whenever the buffer is full, until the writer consumes the records

## Templates

### DTO

public abstract class SampleDTO {  
 protected long timestamp; protected float a;  
 protected float b;  
 protected String c;  
 protected String d;  
}

### Schema

message sample {  
 required int64 timestamp;  
 required binary binary\_field;  
 required int32 int32\_field;  
 required int64 int64\_field;  
 required boolean boolean\_field;  
 required float float\_field;  
 required double double\_field;  
 required int96 int96\_field;  
}

## API

public interface PartitionWriter<DTO\_WITH\_TIMESTAMP> {  
 public void write(DTO\_WITH\_TIMESTAMP rec) throws IOException;  
}

# Folder Hierarchy

.

├── doc

│   ├── developer\_guide.docx

│   ├── flow.drawio

│   ├── flow.png

├── generator.iml

├── pom.xml

├── spark-warehouse

├── src

│   ├── main

│   │   ├── java

│   │   │   └── ind

│   │   │   └── rd

│   │   │   └── parquet

│   │   │   ├── PartitionedParquetGenerator.java

│   │   │   ├── exception

│   │   │   │   └── SchemaValidationException.java

│   │   │   ├── generator

│   │   │   │   └── DataGenerator.java

│   │   │   ├── util

│   │   │   │   ├── Defaults.java

│   │   │   │   ├── PartitionHelper.java

│   │   │   │   ├── PartitionedParquetWriter.java

│   │   │   │   └── SchemaHelper.java

│   │   │   └── writer

│   │   │   ├── DataWriter.java

│   │   │   ├── PartitionWriter.java

│   │   │   └── impl

│   │   │   └── PartitionWriterImpl.java

│   │   └── resources

│   │   └── log4j.properties

│   └── test

│   └── java

│   └── ind

│   └── rd

│   └── parquet

│   └── generator

│   ├── DataGeneratorTest.java

# Usage

## Input

Git clone: ???

Maven compile: mvn compile

Run: mvn exec:java -Dexec.mainClass="ind.rd.parquet.PartitionedParquetGenerator" -Dexec.args="-i /tmp/sample.schema"

Input parameters;

-i,--inputSchema <arg> Input message schema

-l,--bufferLimit <arg> Number of records to buffer

-m,--memoryLimit <arg> Maximum memory buffer while partitioning

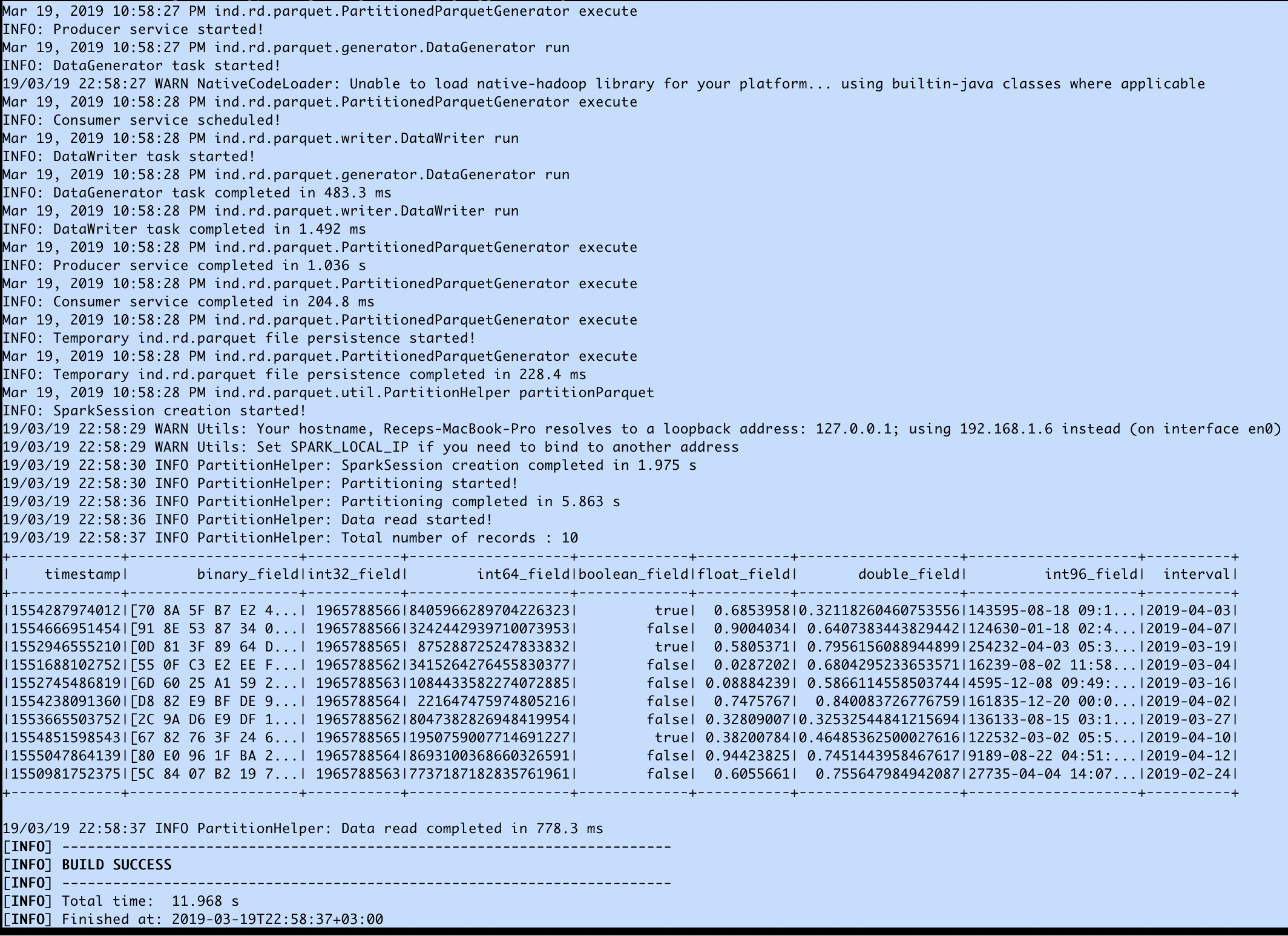
-o,--targetFileName <arg> Output file name

-r,--numberOfRecords <arg> Number of records to generate

-s,--partitionInterval <arg> Partitioning time interval as hours

-t,--threadCount <arg> Number of threads

## Console Output



## File System Output

