# Time Series Processing in JCuda

## Problem

Time series have no standard representation in software tools, neither in real world. One has to distinguish continuous time series (usually discrete in time and values) and event time series (also called event series).

A variety of tools have been developed in multiple languages in order to deal with time series data.

|  |  |
| --- | --- |
| Repo | Description |
| https://github.com/jrachiele/java-timeseries |  |
|  |  |
|  |  |

This project aims on a simplification and smooth integration of time series processing tools in multiple environments, especially on Spark.

## The underlying assumptions are:

* Java 1.8 is used for development
* Each time series is represented as *TimeSeriesObject* and persisted as Avro-Record.
* Multiple time series are represented as *TimeSeriesBucket*. Such a bucket – it is simply a Java class – offers functionality to work on the contained time series row by row or even in a bulk.
* A *TimeSeriesPairBucket* is provided for correlation analysis.
* TimeSerieBucketProcessingContext offers an interface to trigger processing on the data in such a bucket.
  + Examle 1: Processing on Spark Executors
    - A large Avro-file gets split into partitions by Spark automatically.
    - Each partition can be used in one TimeSeriesBucket.
    - The *TimeSeriesBucket* object offers iteration for one by one processing, pair iterators for processing all time series pairs in this bucket and also tuple iterators for working on all tuples of time series in this bucket inside Spark Executors.
    - Furthermore, the content of the bucket can be pushed into a GPU one by one or even completely if the data set is not too large.
    - Single record processing in a GPU makes the most sense for Fluctuation analysis and FFT (experiments and benchmark will be provided in this project).
    - Full bucket upload is useful for fast creation of correlation networks combined with network analysis on time dependent networks.
      * Note: This approach starts with a list of time series, processes episode from all time series pairs to get a correlation matrix per time interval just to calculate the topological property of this matrix over time.
      * By loading all series of a bucket into the GPU’s RAM only once and obtaining all relevant topological properties at the end for the full bucket we have the smallest overhead for data transfer.

## Identify Workloads for Spark and GPU:

Preparation of an appropriate time series set is a task for the Hadoop / Spark cluster. Once, the time series are created and stored as TimeSeriesBucket, e.g., using a parquet file with Avro serialization it is straight forward to process the content in a variety of methods on Spark cluster. The limit is given by Executor RAM and DISK space in this case. HDFS offers huge capacities for time series stores of any size.

To pre-process the data and to specify smaller problems we use the collocated scalable processing engines Impala or Spark on top of the scalable storage layer. This fast in place aggregation, e.g., in case of time series stores implemented on Kudu or HBase, allows a separation of workloads. Not every problem must be handled by a GPU – but specific problems have a huge benefit (see our experiments).

## Related Work

The H2O libraries can be used inside the SparkContext on an EDH cluster via “SparklingWaterApps”. This blog article from 2015 illustrates the approach: <http://blog.cloudera.com/blog/2015/10/how-to-build-a-machine-learning-app-using-sparkling-water-and-apache-spark/>

Figure 1 and Figure 2 can be seen as architectural guide in this project.   
Figure 3 illustrates the advantage of combining AI procedures with traditional workloads such as ETL, analytics, and machine learning.

MapR-DB

https://mapr.com/blog/loading-time-series-database-100-million-points-second/