HOBBIT

Collaborative Project

Holistic Benchmarking of Big Linked Data

Project Number: 688227 **Start Date of Project:** 2015/12/01 **Duration:** 36 months

Deliverable 4.2.1
First Version of Analytics Benchmark

Dissemination Level	Public
Due Date of Deliverable	Month 18, 31/05/2017
Actual Submission Date	Month 18, 31/05/2017
Work Package	WP4 - Analytics Benchmark
Task	T4.2
Type	Demonstrator
Approval Status	Final
Version	1.0
Number of Pages	17

Abstract: This deliverable presents the first version of the data analytics benchmark for HOBBIT Platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688227

History

Version	Date	Reason	Revised by
0.1	09/04/2017	First draft	Roman Katerinenko
0.2	19/05/2017	Final version	Roman Katerinenko

Author List

Organization	Name	Contact Information
AGT International	Roman Katerinenko	rkaterinenko@agtinternational.com
AGT International	Martin Strohbach	mstrohbach@agtinternational.com

Executive Summary

This document describes the first version of the data analytics benchmark. It is based on streaming data anomaly detection scenario which is one of the typical scenarios for analytics applications, especially for Industry 4.0 domain. The benchmark covers benchmarking of unsupervised machine learning methods by incorporating fundamental concepts of this domain: Markov Model and K-means clustering.

Contents

1 Introduction

There is a lot of work in progress in the benchmarking field, especially in the benchmarking of analytics applications. Typically, the notion of analytics application is associated with data, meaning that the aim of such an application is to produce a useful interpretation of the raw data. Data can be divide into two classes each of which requires different analytics techniques. The first class covers structured data. For example, tables or files with a regular format. The second class is the opposite. It represents data which lacks any regular structure. There are two examples of data of this type — software logs and natural language text.

The hallmark of the analytics applications is a mathematical model that allows to analyze the data and make insights. Nowadays, the majority of such models are machine learning models, either supervised or unsupervised. The first version of our benchmark described in this document covers analytics on structured data and unsupervised machine learning models, specifically, the one based on K-means clustering. This is exactly the purpose of task 4.2 — structured machine learning Structured Machine Learning benchmark.

It is a well-known fact among analytics experts that realistic data generation is part of every benchmarking proposal, as it is practically impossible to evaluate analytics-application without a workload based on realistic data. Usually, it is not possible to get a huge amount of data from social websites and other organizations like smart grid industry due to data privacy issues, so realistic data generation should be part of every benchmarking platform. Furthermore, a realistic data generator will allow forming different workloads based on the configurations.

Therefore, the goal of this benchmark is to address the following issues:

- Develop a benchmark based on the task that is representative for analytics software.
- Develop a data generator simulating realistic data for analytics domain.

Tabel 1 shows our open-source contribution to the project:

Project	URL	Description
Benchmark	https://github.com/hobbit-project/sml-benchmark	Benchmark itself including data generator and anomaly detector modules
Data generator	https://github.com/hobbit-project/mm-datagen	Molding machine data streams generator

Table 1: List of open source projects created during T4.2.

This document is structured in the following way. The following Section 2 describes benchmark in detail: architecture, metrics, output data, configuration parameters. It is followed by Section 3 describing data generator: mimicking original data, generation approach, data formats, configuration parameters and data generator properties. Section 4 describes typical analytics scenario of anomaly detection and our implementation which serves as a "Gold standard" for the benchmark. Finally, Section 5 presents our experience of using our benchmark and HOBBIT Platform for Grand Challenge of The 11th ACM International Conference on Distributed and Event-Based Systems (DEBS GC).

2 Analytics Benchmark

The benchmark is performed in a streaming fashion. That is, the benchmark is able to compute metrics, generate data, evaluate benchmarked system output at the same time. In this sections, we will describe in detail key elements of such a streaming workflow.

2.1 Benchmark Architecture

Main benchmark components are described below.
Benchmark Controller. The Benchmark Controller is the main entry point of the benchmark. It mostly responsible for communication with the platform. Specifically, it provides configuration parameters given by the platform controller to internal modules and sends task's result back to the platform controller. Besides that, it executes tasks and is able to stop them by time out.
Benchmark Task. Conceptually, a task is an entity which represents work-flow of one benchmark stage. In the current version of the benchmark, we have only one task representing "Anomaly detection scenario" described later. This scenario requires orchestration of four following components: Data Generator, Anomaly Detector, Data Checker, Benchmark-side protocol.
Data Generator. This component is responsible for generation realistic workload. It is a complex task described in detail in Section 3 for details.
Anomaly Detector. The purpose of this component is to produce "gold standard" which is used to match benchmarked system output against. The gold standard is a stream of anomaly description where the description contains all needed information to relate anomaly to a machine, machine's sensor, time and probability of the anomaly. See Section 4 for more details about anomaly detection.
Data Checker. Having the gold standard and the output from the benchmarked system, this component performs a matching operation.
System-side and benchmark-side protocols. Since benchmark and benchmarked system are running asynchronously, they need to follow a certain synchronization protocol that allows them to communicate properly. Two parts of the protocol are encapsulated in Benchmark-side and System-side components respectively. System-side protocol simplifies integration of the benchmark and the benchmarked system because the developer of the latter doesn't need to know the protocol bur merely use our component.
Timer. The purpose of this component is to fire time out signal and terminate benchmark.
Communication. This component encapsulates a mean of communication. In our benchmark, we have two implementations: in-memory and network-based.

Figure 1 pictures dependencies among the components while Figure 2 shows data flow within the benchmark. The latter depicts that Data Generator uses Communication to send generated data to the benchmarked system. And at the same time it sends the same data to Anomaly Detector. Anomaly Detector produces "Gold standard" anomaly delivered to Data Checker which matches "Gold standard" anomaly to the system output and calculates latency. Latency is the time gap between the instant when last data point contributing to anomaly has been sent to the system and the time when the anomaly was received from the system.

2.2 Workload

The benchmark is able to generate configurable workload at execution time. The workload is a stream of data points delivered to the benchmarked system using standard interface of the HOBBIT Platform. Each data point is a vector of simulated sensor measurements from one molding machine. For more

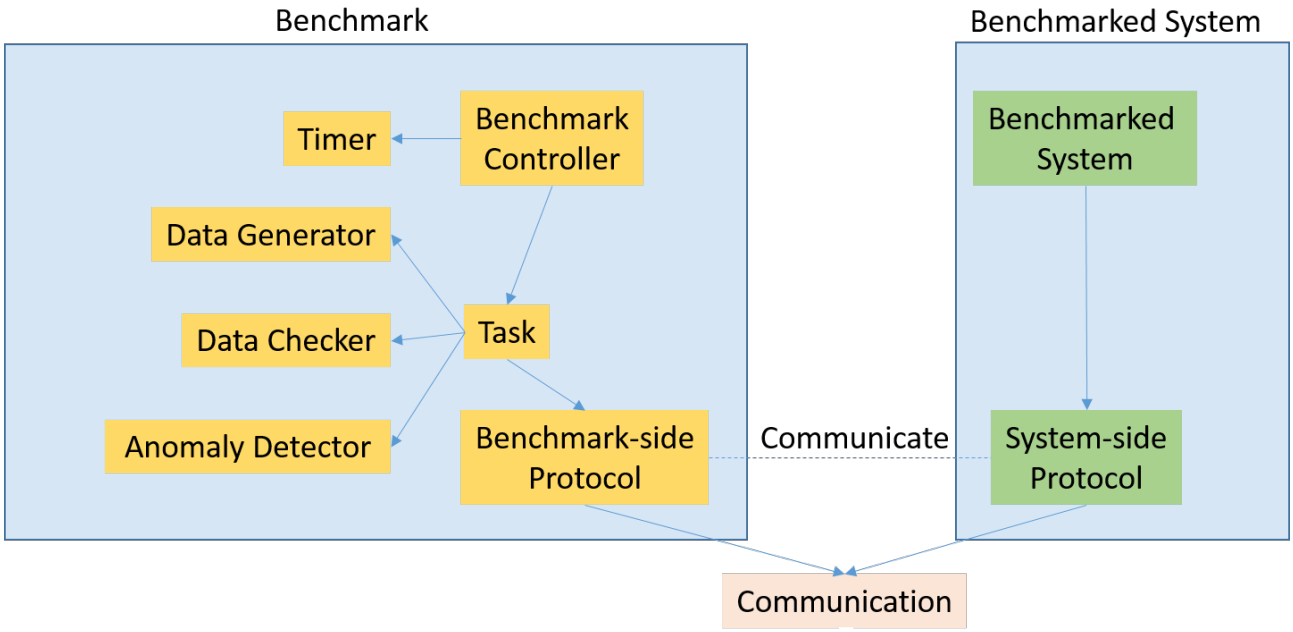

Figure 1: Benchmark Architecture (module dependencies)

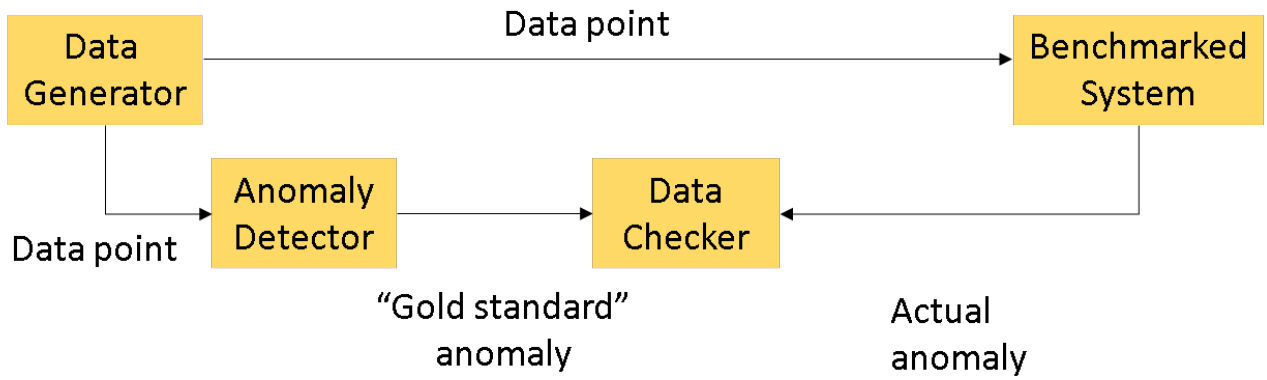

Figure 2: Flow of data

details regarding data generation refer to Section 3.4.

The benchmark is very flexible and suitable for different types of analytics. It is able to output data points in CSV or RDF(N-Triples) formats and supports two streaming strategies that can be configured using user interface (GUI) of the HOBBIT Platform:

- **Fixed amount of machines.** This strategy simulates fixed amount of molding machine streaming data.
- **Increasing amount of machines.** This strategy starts streaming data from the given initial amount of machines and introduces a new machine after every X data points, where X is a

configurable parameter.

2.3 Benchmark Metrics

The benchmark is able to measure 3 common for analytics applications metrics:

- **Accuracy.** The Amount of anomalies matched against gold standard.
- **Latency.** Latency is the amount of time between the last data point contributing to the anomaly and the time when the anomaly was received from the benchmarked system.
- **Throughput**. An average amount of bytes being processed by the system per second.

Figure 3 depicts screenshot of HOBBIT Platform user interface with the result of execution of our benchmark on our benchmarked system.

Parameter	1493032425815
Benchmark	DEBS GC benchmark
System	DEBS system csv correct
Challenge Task	
Error	
http://www.debs2017.org/gc/debsKpi	Executed on 63 data points, interval 10 millis, checked 6 anomalies, result: Anomalies matched successfully
http://www.debs2017.org/gc/averageLatencyNanos	12000000
http://www.debs2017.org/gc/throughputBytesPerSecond	75818.134981968061766
http://www.debs2017.org/gc/terminationType	Terminated correctly.

Figure 3: Benchmark metrics in the HOBBIT Platform

2.4 Benchmark Configuration

The following parameters can be set to the benchmark using HOBBIT Platform (See Figure 4). The list can be logically broken down into two parts: performance-related and accuracy-related (or domain-specific). Accuracy-related configuration:

- **Window size.** The window of this size is moved over the data stream, one data point at a time. Each move triggers anomaly detection.
- **Transitions count.** It specifies the length of data point sequence to be checked for an abnormal probability. Probability is deemed abnormal if its value is below a given threshold.
- **Max. clustering iterations.** Maximum amount of clustering iterations for K-means algorithm which is used for anomaly detection.
- **Probability threshold.** The sequence of data points having probability threshold below given is considered abnormal.

Performance-related configuration:

- **Machine Count.** Amount of molding machines to simulate.

- **Seed.** A number to initialize random generator.
- **Amount of messages.** A number of messages (data points) to be generated for one machine.
- **Interval between measurements. (nanos.)** Specifies sending messages interval.
- **Output format.** Two possible options are RDF and CSV.
- **Benchmark mode.** There are two modes. The first mode is the default one when fixed amount of machines being generated. The second mode is when increasing amount of machines being generated.
- **Timeout.** Number of minutes to stop benchmark after its start.

3 Data Generator

The data generator we have implemented in this work-package is able to mimic sensor data coming from production molding machines. It based on real data obtained through cooperation with Weidmueller. The data generator is able to produce streams of sensor measurements. Each measurement (or "data point") consists of 120 values. Such a workload can be used for many scenarios. To give an example, data ingestion, analytics accuracy, analytics performance. The fact that the data simulates real-world sensor data, makes it useful in other domains too. For instance, Internet of Things, Industry 4.0.

Data generation process consists of two main steps:

1. Model calculation
2. Data generation

Once the model has been computed using the original sample as an input, it can be serialized and used further without a need to keep the original sample. Therefore, it allows protecting the confidentiality of original data sample. All these steps are described below in detail.

3.1 Original Sample

Original sample has the following properties:

- Contains real sensor measurements from production injection molding machine
- Contains around 17000 data points
- Each data point is 120-dimensional vector of measurements
- Measurements are of several formats: date, time, integer, decimal, text
- Measurements represent diverse physical-world values: distance, pressure, time, frequency, volume, temperature, time, speed, force

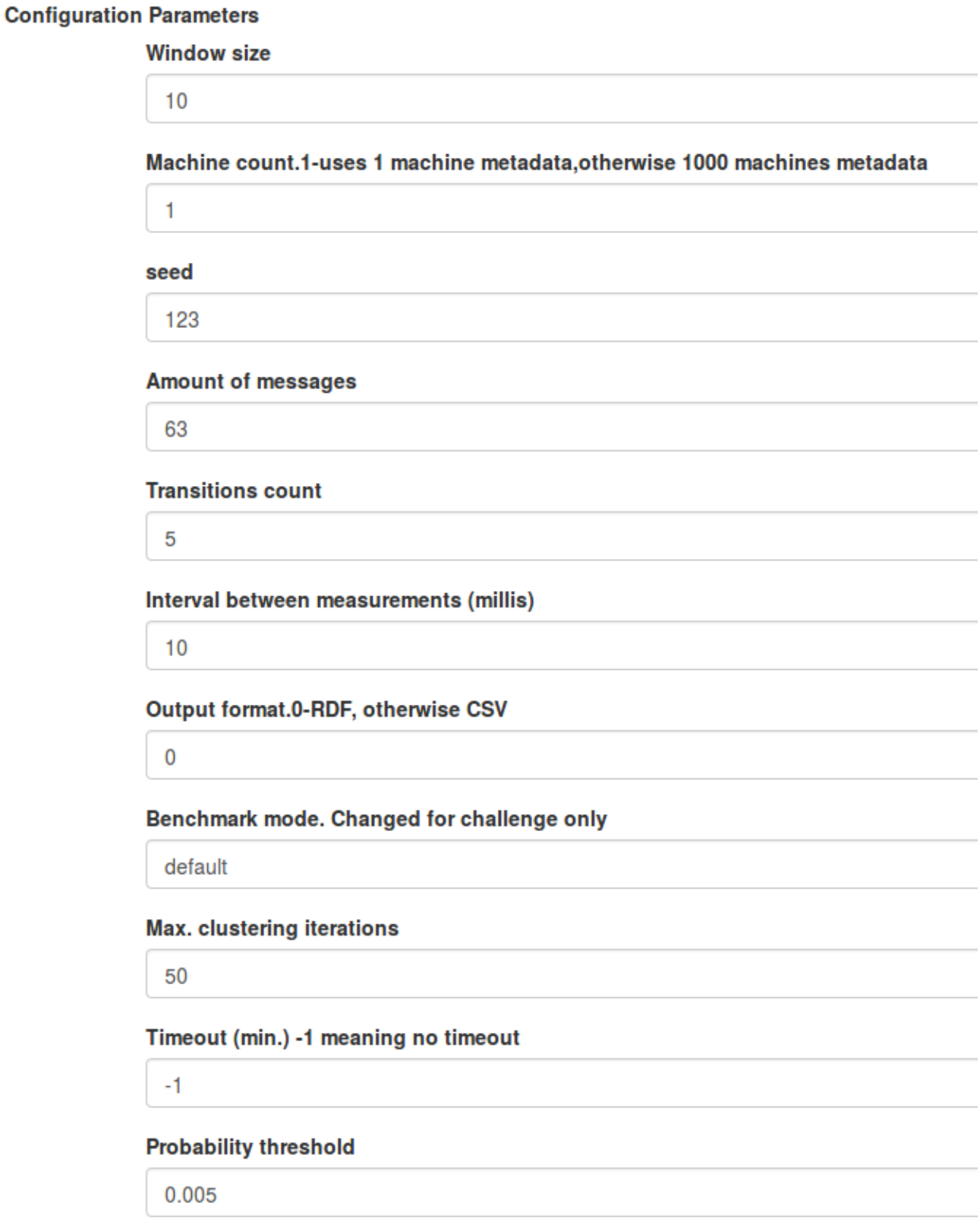

Figure 4: Benchmark parameters in the HOBBIT Platform

3.2 Dimensions Classification

Working with the sample we have noticed that in order to simulate it more precisely we need to use different techniques for different dimensions. Therefore, we do dimensions classification as a pre-processing for model calculation step. The classification is empirical. We have implemented a module which is able to classify dimensions by the following classes automatically.

- **Text dimension.** This dimension has only text values.
- **Constant dimension.** This dimension has only one value for all data points.
- **Date dimension.** Values of this dimension represent date when the corresponding measurement was taken.
- **Time dimension.** Values of this dimension represent a time when the corresponding measurement was taken.
- **"Trending phase" dimension.** This dimension contains periods when values are monotonically increasing or decreasing.
- **"Stateful" dimension.** It is called "stateful" because the values of this dimension show how the molding machine goes through different states as the time goes.

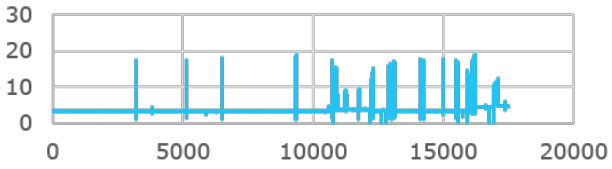

Figure 5: Example of a "stateful" dimension

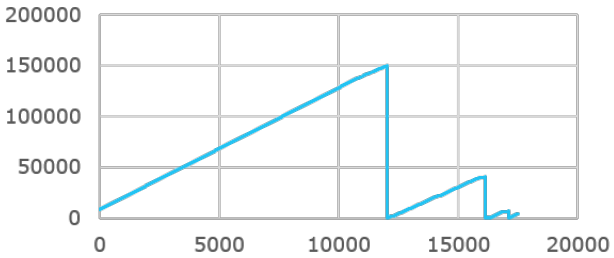

Figure 6: Example of "trending phase" dimension

The classification module uses a state machine to iterate over dimension values and compute what is called a "flatness" function. The value of flatness function is then used to classify dimensions and to derive the amount of clusters for stateful dimensions. Simply speaking, flatness is equal to the total length of flat areas divided by the total length of non-flat areas. We call area "flat" if a dimension has the same values consequently in that area. For constant dimension flatness is 1, for strictly ascending/descending dimensions flatness is 0. We compute F based on flatness in order to estimate the amount of clusters on further stages. The motivation for F is the following. Given that K is the parameter associated with the max. amount of clusters, we want F to return smaller clusters count for more flat dimensions and bigger clusters count for the others. Maximum should be K.

$$F = max(((1 - flatness + 0.05) * K), 1)$$

This classification is used on the next steps during model computation and data generation.

3.3 Computing Mathematical Model of the Sample

Mathematical characteristics computed for a dimension depends on its type. They are obvious for most of the dimensions, but for trending phase and stateful types they are non-trivial and deserve detailed explanation provided below.

3.3.1 "Stateful" Dimension Model Computation

After the dimension has been classified as a stateful we compute amount of distinct values D. We use D to derive the amount of clusters C that will be used for this dimension. After that, we use K-means clustering to produce up to C clusters. For each cluster, we compute mean and standard deviation values. Then, we iterate over dimension data points and compute one–step transition probability model (Markov Model) for clusters associated with data points. Markov Model, mean, standard deviation, distinct values count from the model of a stateful dimension.

3.3.2 "Trending Phase" Dimension Model Computation

For this type of dimensions we compute and save the following values:

- First index where first significant value change occurs
- Average increment/decrement for monotonic phases

These values form a statistical model of a trending phase dimension.

3.4 Data Generation

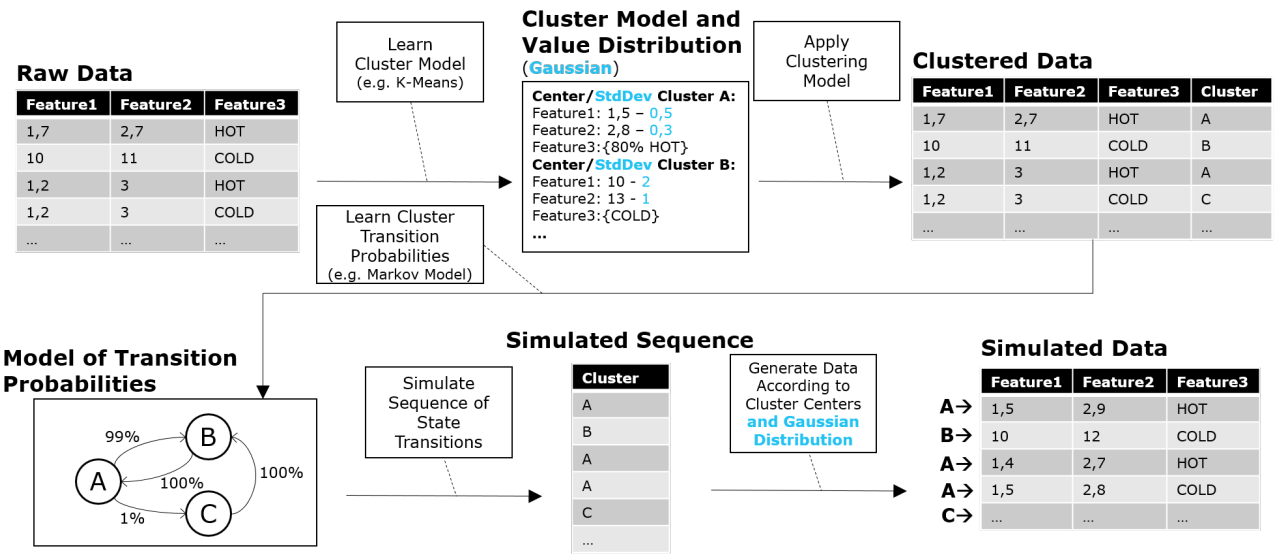

Figure 7: Mimikcing stateful dimension

In data generation phase we use the model of each original dimension to mimic original values. Each dimension type is treated with a different algorithm. Most of them are simple and don't deserve attention. For instance, in order to produce next value of date dimension we add the certain interval to the chosen starting date. Stateful dimension generation is different and we will explain that in detail. Figure 7 shows the steps of the algorithm. After the model of a stateful dimension has been computed we walk through Markov Model in a random order. For each node we have visited we generate a value:

$$value = randomGaussianValue * standardDeviation + mean$$

That allows us to produce an infinite sequence of values that mimic original values in a statistical sense. Example of original and simulated plots are shown on figure 8 and 9

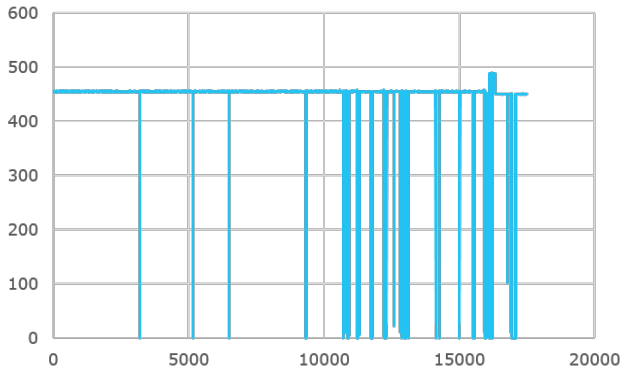

Figure 8: Stateful dimension-original values

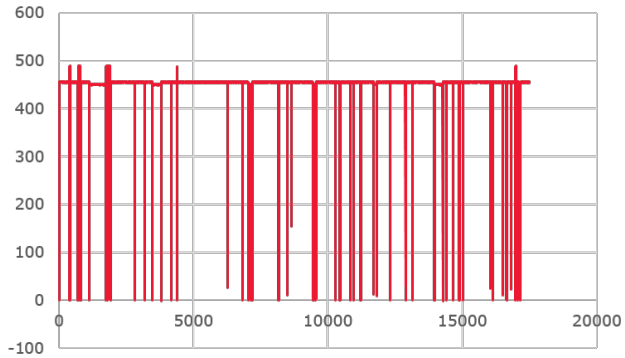

Figure 9: Stateful dimension-simulated values

3.5 Data Generator Configuration

The Data Generator has highly configurable API. The following parameters can be tuned:

- Amount of molding machines and needed data points count
- Time interval between data points
- Seed for random generator. The same seed value results in the same data points being produced.

- Generation frequency
- Output encoding
- Output data format: CSV or RDF (NTriples).

3.6 Output Formats

Our data generator is able to format generated data in 2 ways: CSV and RDF(NTriples). CSV output consists of similarly-formatted rows. Each row represents one data point. Data point consists of measurements separated by a comma. The total number of measurements constituting one data point is 120.

RDF encoding is based on semantic ontology. The graph on Figure 10 depicts an example of observations as they will be sent over the stream according to the ontology. All observations are grouped in an observation group. The observation group links to the machine identifier $wmm : MoldingMachine_1$that links the observation to the corresponding metadata. The observation group contains a timestamp ($ssn : observationResultTime$) and the injection cycle ($debs : Cycle_2$) to which all observations relate. The actual readings can be obtained by traversing the $i40 : contains/ssn : observationResult/ssn : hasValue/IoTCore : valueLiteral$ property path. Note that in the actual data there may be more observations.

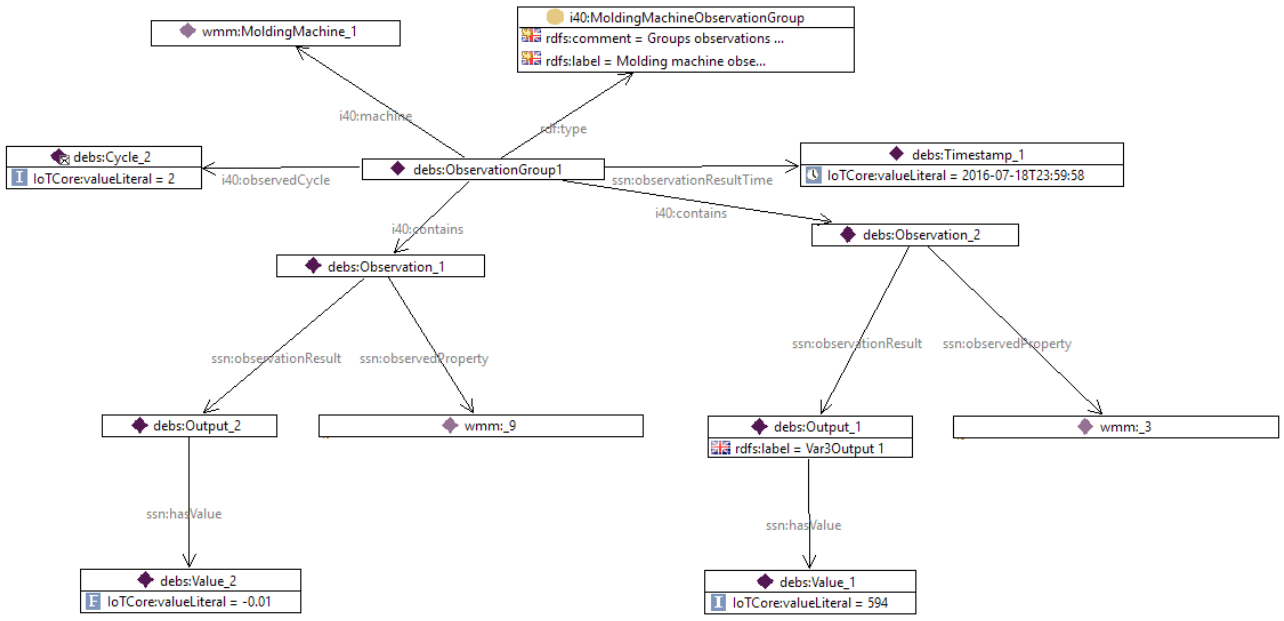

Figure 10: Example of data point formatted as RDF

Below is an example of serialized triples as they could be streamed:

```
1 debs:ObservationGroup_1 rdf:type i40:MoldingMachineObservationGroup.
2 debs:ObservationGroup_1 ssn:observationResultTime debs:Timestamp_1.
3 debs:ObservationGroup_1 i40:contains debs:Observation_1.
4 debs:ObservationGroup_1 i40:contains debs:Observation_2.
5 debs:ObservationGroup_1 i40:observedCycle debs:Cycle_2.
6 debs:ObservationGroup_1 i40:machine wmm:MoldingMachine_1.
7 debs:Cycle_2 rdf:type i40:Cycle.
```

```
8 debs:Cycle_2 IoTCore:valueLiteral "2"^^xsd:int.
9 debs:Timestamp_1 rdf:type IoTCore:Timestamp.
10 debs:Timestamp_1 IoTCore:valueLiteral "2016-07-18T23:59:58"^^xsd:
    dateTime.
11 debs:Timestamp_1 rdfs:label "Timestamp 20160718 23 59 58"@en.
12 debs:Observation_1 rdf:type i40:MoldingMachineObservation.
13 debs:Observation_1 ssn:observationResult debs:Output_1.
14 debs:Observation_1 ssn:observedProperty wmm:_3.
15 debs:Output_1 rdf:type ssn:SensorOutput.
16 debs:Output_1 ssn:hasValue debs:Value_1.
17 debs:Value_1 rdf:type i40:NumberValue.
18 debs:Value_1 IoTCore:valueLiteral "594"^^xsd:int.
19 debs:Observation_2 rdf:type i40:MoldingMachineObservation.
20 debs:Observation_2 ssn:observedProperty wmm:_9.
21 debs:Observation_2 ssn:observationResult debs:Output_2.
22 debs:Output_2 rdf:type ssn:SensorOutput.
23 debs:Output_2 ssn:hasValue debs:Value_2.
24 debs:Value_2 rdf:type i40:NumberValue.
25 debs:Value_2 IoTCore:valueLiteral "-0.01"^^xsd:float.
```

Figures 11 and 12 show in detail relation between two encodings. The row at the top of pictures depicts comma-separated values of CSV. Related RDF-triples are shown below the row.

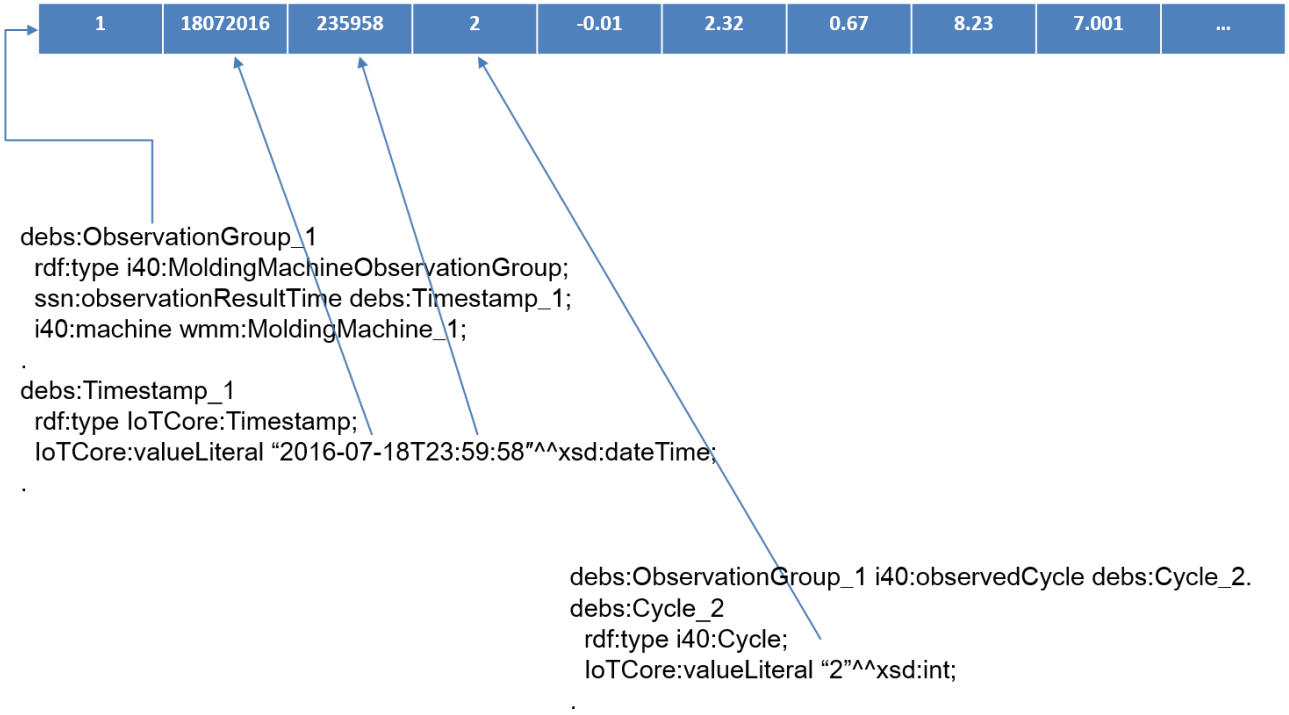

Figure 11: Relation between one CSV row and its RDF encoding

3.7 Properties of the Data Generator

Our data generator has the following main properties:

- It mimics original molding machine dataset but it doesn't copy it. That is, it preserves the confidentiality of the original data.

1	18072016	235958	2	-0.01	2.32	0.67	8.23	7.001	...

```
debs:ObservationGroup_1 i40:contains debs:Observation_1.

debs:Observation_1
  rdf:type i40:MoldingMachineObservation;
  ssn:observationResult debs:Output_2;
  ssn:observedProperty wmm:_0;
.
debs:Output_2
 rdf:type ssn:SensorOutput;
 ssn:hasValue debs:Value_2;
.
debs:Value_2
 rdf:type i40:NumberValue;
 IoTCore:valueLiteral "-0.01"^^xsd:float;
.
```

Figure 12: Encoding of one value in RDF

- It allows to scale a small sample up, which can be used for performance testing and Big Data systems testing under a realistic workload.
- It mimics the states each dimension go through time, but it loses correlation among dimensions.
- It is well suited for benchmarking typical analytics tasks. E.g., anomaly detection.
- It outputs RDF data format which can be used to benchmark Semantic Technologies.

4 Anomaly Detector

For the first version of this benchmark, we have chosen an anomaly detection use-case as one of the typical and representative tasks for modern analytics applications. Besides that, the anomaly detection algorithm we developed has another important property — it uses simple and well–known concepts which are basic building blocks of most analytics applications: Markov Model, K-means clustering, mean and standard deviation.

The data is clustered and the state transitions between the observed clusters are modeled as a Markov chain. Based on this classification, anomalies are detected as sequences of transitions that happen with a probability lower than a given threshold. More specifically, The query has three stages: (1) Finding Clusters, (2) Training a Markov Model and (3) Finding Anomalies. Figure 13 illustrates the query stages as NFA. Once started, the activities for each stage are executed continuously and never stop, e.g., cluster centers are continuously evaluated while the Markov model is already used for anomaly detection.

An event passes the sketched stages in sequence. This means that a changed cluster center must be considered in the subsequent stages right after the centers have changed. An event that causes a change of a cluster center first causes the update of the centers, then an update of the Markov model and is finally used in anomaly detection.

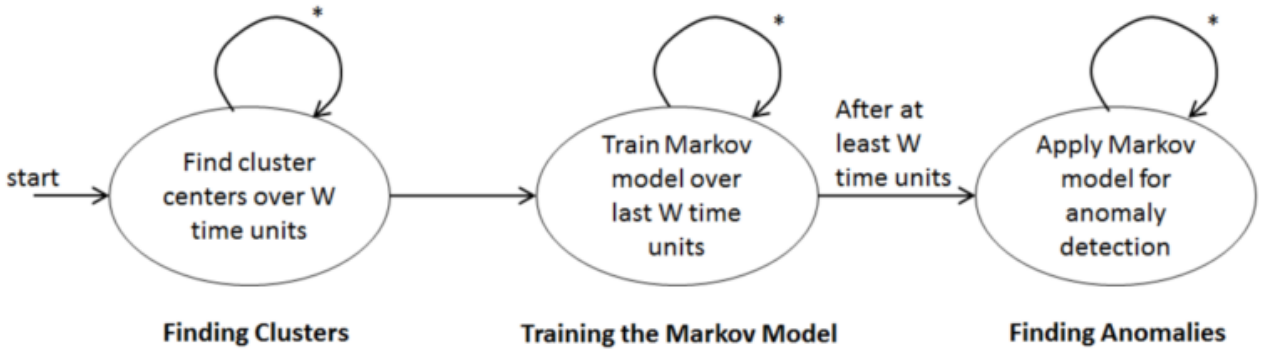

Figure 13: Stages of anomaly detection algorithm

Finding anomalies consists of the following steps:

1. For each stateful dimension, find up to and maintain K cluster centers, using the numbers 1 to K as seeds for the initial K centroids. The number K is defined in the metadata for each dimension of each individual machine. Use all measurements from the last W time units to find the cluster centers.
2. The initial cluster centers for each dimension of measurements are determined by the first K distinct values for that dimension in the stream.
3. When recomputing the clusters after shifting the time window, the cluster centers are determined by the first K distinct values for that dimension in the given window.
4. If a given window has less than K distinct values than the number of clusters to be computed must be equal to the number of distinct values in the window.
5. If a data point has the exact same distance to more than one cluster center, it must be associated with the cluster that has the highest center value.
6. Determine the transition probabilities by maintaining the count of transitions between all states in the last W time units. For determining a transition at time t, use the cluster centers that are valid at time t, i.e., no remapping of past observations to clusters in retrospect is required.
7. Output an alert about a machine, if any sequence of up to N state transitions for that machine is observed that has a probability below T.

5 Using Benchmark in the ACM DEBS Grand Challenge

As a part of our activities in the HOBBIT project, we co-organized ACM DEBS Grand Challenge where our benchmark and HOBBIT platform were main elements of the challenge: http://www.debs2017.org/call-for-grand-challenge-solutions/.

The 2017 ACM DEBS Grand Challenge is the seventh in a series of challenges aimed to evaluate both research and industrial event-based systems for real-time analytics. This year focus is on high velocity and high volume data streams. Both the data set and the automated evaluation platform are provided by the HOBBIT project. This allows offering the possibility of running a distributed solution on multiple virtual machines.

Challenge task is aimed at implementation of previously mentioned anomaly detection algorithm in the most efficient way. Our benchmark streams data to the benchmarked system with generated data and checked latency, throughput, the correctness of found anomalies.

At the moment of writing this deliverable, the challenge was in progress. It has attracted more that 15 participants organized in teams. Seven teams managed to pass the first stage of challenge and approved for the final evaluation. The number of teams includes at least one commercial system.

5.1 Tutorial Benchmark and System

In order to demonstrate integration with the HOBBIT platform API for challenge–takers we implemented a tutorial benchmark and tutorial system. Source code for both can be found on HOBBIT source–code site https://github.com/hobbit-project. Figure 14 shows our artifacts uploaded to the HOBBIT Platform. There are two benchmarks: analytics benchmark and tutorial benchmark. For each benchmark, we have developed two systems. The one which can pass corresponding benchmark and the one that fails in a planned way. This way we demonstrate that the benchmarks are able to detect both positive and negative results.

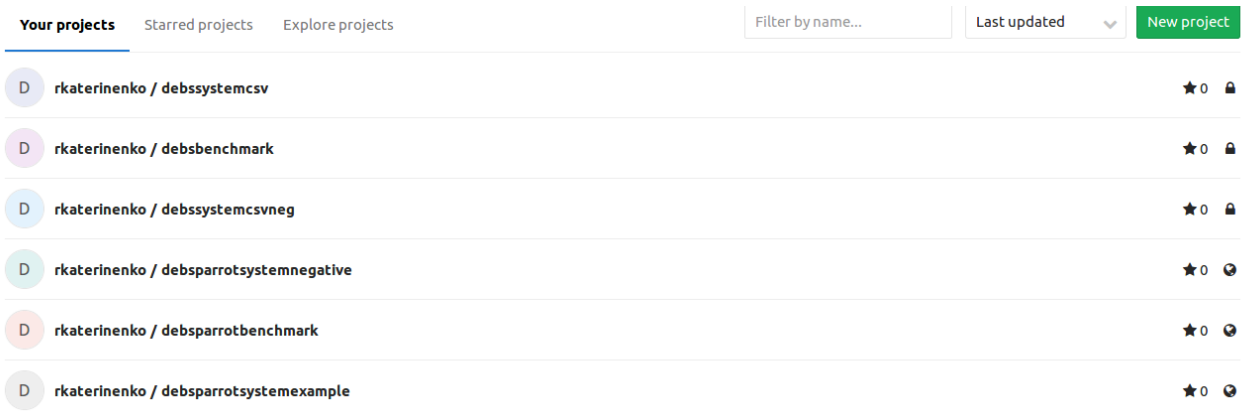

Figure 14: Our benchmarks and system uploaded to the HOBBIT Platform

6 Summary

The first version of the data analytics benchmark described in this document allows to generate and stream sensor data. Its flexible configuration allows producing load suitable for benchmarking variety of analytics software. The benchmark is based on representative use-case from analytics systems domain — anomaly detection. The benchmark and HOBBIT Platform was chosen ACM DEBS Grand Challenge organizers as a main system for the challenge, which shows that the benchmark adequately addresses benchmarking tasks in analytics domain.