Collaborative Project
Holistic Benchmarking of Big Linked Data
Project Number: 688227       Start Date of Project: 2015/12/01       Duration: 36 months

Deliverable 3.1.1
Data Extraction Benchmark for Sensor Data

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>Public</th>
</tr>
</thead>
<tbody>
<tr>
<td>Due Date of Deliverable</td>
<td>Month 18, 31/05/2017</td>
</tr>
<tr>
<td>Actual Submission Date</td>
<td>Month 18, 31/05/2017</td>
</tr>
<tr>
<td>Work Package</td>
<td>WP3 - Benchmarks I: Generation and Acquisition</td>
</tr>
<tr>
<td>Task</td>
<td>T3.1</td>
</tr>
<tr>
<td>Type</td>
<td>Report</td>
</tr>
<tr>
<td>Approval Status</td>
<td>Final</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Number of Pages</td>
<td>38</td>
</tr>
</tbody>
</table>

Abstract: This deliverable presents the first version of the ODIN (StOrage and Data Insertion beNchmark) benchmark for storage and retrieval of streamed data for the HOBBIT platform.

The information in this document reflects only the author’s views and the European Commission is not liable for any use that may be made of the information contained therein. The information in this document is provided "as is" without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/her sole risk and liability.

Project funded by the European Commission’s Horizon 2020 Program.
History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>26/04/2017</td>
<td>First draft created</td>
<td>Kleanthi Georgala</td>
</tr>
<tr>
<td>0.2</td>
<td>02/05/2017</td>
<td>First revision</td>
<td>Kleanthi Georgala</td>
</tr>
<tr>
<td>0.3</td>
<td>02/05/2017</td>
<td>First review</td>
<td>Giulio Napolitano</td>
</tr>
<tr>
<td>0.4</td>
<td>15/05/2017</td>
<td>Reviewed Version</td>
<td>Kleanthi Georgala</td>
</tr>
<tr>
<td>0.9</td>
<td>29/05/2017</td>
<td>Second review</td>
<td>Axel Ngonga</td>
</tr>
<tr>
<td>1.0</td>
<td>15/05/2017</td>
<td>Final</td>
<td>Kleanthi Georgala</td>
</tr>
</tbody>
</table>

Author List

<table>
<thead>
<tr>
<th>Organization</th>
<th>Name</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfAI</td>
<td>Kleanthi Georgala</td>
<td><a href="mailto:georgala@informatik.uni-leipzig.de">georgala@informatik.uni-leipzig.de</a></td>
</tr>
</tbody>
</table>
Executive Summary

This document describes the first version of the ODIN benchmark for data extraction solutions for structured data. The benchmark is designed to evaluate the backend of these solutions (especially the acquisition phase) by simulating the ingestion, storage and retrieval of streams of RDF data. To this end, ODIN emulates loads faced by triple store during the insertion of triples by an extraction solution for enterprise data (e.g., industry sensors) based on models derived from real data. The key performance indicators during the evaluation are completeness and efficiency.
Contents

1 Introduction 8

2 ODIN Benchmark 8
   2.1 ODIN Parameters .................................................. 8
   2.2 Components ......................................................... 10
      2.2.1 Benchmark Controller ......................................... 10
      2.2.2 Data Generator ................................................ 10
      2.2.3 Task Generator ............................................... 10
      2.2.4 System Adapter ................................................ 11
      2.2.5 Evaluation Module .......................................... 11
   2.3 Data Pre-processing and Streams .............................. 11

3 Approach 13
   3.1 Initialization phase .............................................. 13
   3.2 Loading phase ..................................................... 14
   3.3 Querying phase ................................................... 15
   3.4 Evaluation Phase .................................................. 15
   3.5 Termination phase ................................................ 16

4 Evaluation 16
   4.1 Baseline System: OpenLink Virtuoso ........................... 16
   4.2 KPIs ................................................................. 16
   4.3 Results ............................................................. 18
      4.3.1 Experiments with different numbers of data generators - agents .......... 18
      4.3.2 Experiments with different population values of generated data ........ 21
      4.3.3 Experiments with different numbers of insert queries per stream ........ 25
      4.3.4 Primarily results ........................................... 28

5 Conclusion and Future Work 38
List of Figures

1 ODIN benchmark parameters ................................................................. 9
2 Data Preprocessing Phase ........................................................................ 15
3 Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1, 2, 4, 8 and 16. .................................................. 19
4 Average Delay of tasks for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1, 2, 4, 8 and 16. .................................................. 20
5 Maximum Triples-per-Second for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1, 2, 4, 8 and 16. The result for 16 DG is not visible in the figure because it is very low compared to the other values (38.65). .................................................. 20
6 Recall, Precision and F-measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1. The horizontal axis represents the number of tasks. .................................................. 21
7 Recall, Precision and F-measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 2. The horizontal axis represents the number of tasks. .................................................. 21
8 Recall, Precision and F-measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 4. The horizontal axis represents the number of tasks. .................................................. 22
9 Recall, Precision and F-measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 8. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...30. The remaining tasks had 0 in all three measurements. .................................................. 22
10 Recall, Precision and F-measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 16. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...40. The remaining tasks had 0 in all three measurements. .................................................. 23
11 Task Delay for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1. The horizontal axis represents the number of tasks. .................................................. 23
12 Task Delay for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 2. The horizontal axis represents the number of tasks. .................................................. 24
13 Task Delay for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 4. The horizontal axis represents the number of tasks. .................................................. 24
14 Task Delay for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 8. The horizontal axis represents the number of tasks. ................................................................. 25
15 Task Delay for Number of insert queries per stream = 100, Population of generated data = 1000 and Number of data generators - agents = 16. The horizontal axis represents the number of tasks. ................................................................. 25
16 Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 1,000, 5,000, 10,000 and 25,000. ................................................................. 26
17 Average Delay of tasks for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 1,000, 5,000, 10,000 and 25,000. ................................................................. 26
18 Maximum Triples-per-Second for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 1,000, 5,000, 10,000 and 25,000. ................................................................. 27
19 Recall, Precision and F-measure for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 5,000. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...70. The remaining tasks had 0 in all three measurements. ................................................................. 27
20 Recall, Precision and F-measure for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 10,000. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...110. The remaining tasks had 0 in all three measurements. ................................................................. 28
21 Recall, Precision and F-measure for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 25,000. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...100. The remaining tasks had 0 in all three measurements. ................................................................. 28
22 Task Delay for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 5,000. The horizontal axis represents the number of tasks. ................................................................. 29
23 Task Delay for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 10,000. The horizontal axis represents the number of tasks. ................................................................. 29
24 Task Delay for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 25,000. The horizontal axis represents the number of tasks. ................................................................. 30
25 Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 100, 500, 1,000 and 5,000. ................................................................. 30
Average Delay of tasks for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 100, 500, 1,000 and 5,000.

Maximum Triples-per-Second for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 100, 500, 1,000 and 5,000.

Recall, Precision and F-measure for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 500. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...30. The remaining tasks had 0 in all three measurements.

Recall, Precision and F-measure for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 1,000. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...40. The remaining tasks had 0 in all three measurements.

Recall, Precision and F-measure for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 5,000. The horizontal axis represents the number of tasks.

Task Delay for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 500. The horizontal axis represents the number of tasks.

Task Delay for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 1,000. The horizontal axis represents the number of tasks.

Task Delay for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 5,000. The horizontal axis represents the number of tasks.

Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.

Average Delay of tasks of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.

Precision of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.

Recall of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.
38 Maximum Triples-per-Second of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4. ................................. 37

39 Task Delay of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4. ................................. 37

List of Tables

1 Frequency ordering of variable-result pairs ........................................ 14
2 Reference set of SELECT SPARQL query ........................................... 14

Listings

1 Example INSERT SPARQL query ....................................................... 13
2 Atomic triple patterns of example INSERT SPARQL query ..................... 13
3 Derived SELECT SPARQL query ....................................................... 13
1 Introduction

The constant growth of the Linked Data Web in velocity and volume has increased the need for triple stores to ingest streams of data and perform queries on this data efficiently. The aim of this workpackage is to benchmark the performance of triple store solutions when faced with streams of data from industrial machinery in terms of efficiency and completeness. Our goal is to measure the performance of triple stores by storing and retrieving RDF data, considering two main choke points:

- Scalability (Data volume): Number of triples per stream and number of streams.
- Time complexity (Data velocity): Number of triples per second.

The data will be generated from one or multiple resources in parallel and will be inserted using SPARQL INSERT queries. This facet of triple stores has (to the best of our knowledge) never been benchmarked before. SELECT SPARQL queries will be used to test the system’s ingestion performance and storage abilities. The components of the benchmark for this task are implemented in Java.

This document is structured as follows: Section 2 describes the components of the benchmark and the pre-processing phase of the data during the initialization phase. Section 3 provides a detailed description of the different phases of the ODIN benchmark. Section 4 includes a presentation of the baseline storage system we benchmarked using ODIN (Section 4.1), a description of KPIs used to evaluate the baseline system (Section 4.2) and the presentation of the results accompanied by a discussion regarding the baseline system performance (Section 4.3). Finally in Section 4.3.4, we present the primarily results of our evaluation using the aforementioned baseline system, a triple store system developed by ONTOS and the commercial version of the baseline system.

2 ODIN Benchmark

2.1 ODIN Parameters

We begin by explaining the parameters of the ODIN system that are needed for executing the benchmark. The parameters are independent of the triple store that will be evaluated. Figure 1 illustrates an example configuration of the ODIN benchmark. The required parameters are:

- **Duration of the benchmark**: The user must determine the duration of the task by assigning a value in milliseconds to the field. The default value is set to 600000 ms. Note that the duration of each experiment, including all the phases described in Section 3, is at most 40 min.

- **Name of mimicking algorithm output folder**: The relative path of the output dataset folder. There is no default value for this parameter.

- **Number of insert queries per stream**: This value is responsible for determining the number of INSERT SPARQL queries after which a SELECT query is performed. The default value is set to 100.

- **Population of generated data**: This value determines the number of events generated by a mimicking algorithm for one Data Generator. Note that this value might not be equal to the number of generated triples. The default value is set to 1000.

---

1 http://ontos.com/
- **Number of data generators - agents**: The number of Data Generators for this experiment. The default value is 2.

- **Name of mimicking algorithm**: The name of the mimicking algorithm to be invoked to generate data. There are two available values: `TRANSPORT_DATA`\(^2\), that invokes the mimicking algorithm developed by iMec for public transport and `TWIG`\(^3\), that invokes the mimicking algorithm developed for workpackage T3.2 for Twitter messages. The default value is `TRANSPORTDATA`.

- **Seed for mimicking algorithm**: The seed value for a mimicking algorithm. The default value is 100.

- **Number of task generators - agents**: The number of Task Generators for this experiment. The default value is 1.

---

**Figure 1: ODIN benchmark parameters**

---

\(^2\)https://github.com/PoDiGG/podigg
\(^3\)https://github.com/AKSW/TWIG
2.2 Components

We continue by giving a detailed description of the main components of our benchmark for Data Ingestion and Retrieval, dubbed ODIN.

2.2.1 Benchmark Controller

The Benchmark Controller (BC) is the most basic component of the benchmark as it initializes all other components and executes the benchmark. Firstly, it receives from the platform the parameters that are required to properly initialize the Data Generators, the Task Generator and the Evaluation Storage components. Once all components are initialized, the BC sends signals to the platform to allow the Data Generators and the Task Generator to start their tasks. Then, it waits until all Data Generators, the Task Generator and the System Adapter have terminated their execution and proceeds in creating the Evaluation Module of the benchmark. Finally, it waits until the Evaluation Module finishes its execution and sends the performance results to be stored.

2.2.2 Data Generator

A Data Generator (DG) is responsible for creating INSERT SPARQL queries to be performed against the triple store. ODIN can create one or more DGs that run independently of each other. Each DG invokes a unique mimicking algorithm instance, that generates a set of RDF triples, from which the INSERT queries are derived. A mimicking algorithm is responsible for generating data that covers 24 hours of production. Each DG takes as input the following parameters:

- the name of the mimicking algorithm.
- the total duration of the benchmark.
- the number of INSERT queries after which a SELECT query will be performed.
- a unique generation seed value that will be sent to the mimicking algorithm, so that the generated data is unique for a particular DG. Each DG creates its own seed by adding the seed value set by the user (Section 2.1) to its unique ID value.
- a population value that describes the number of unique entities to be generated by the mimicking algorithm

Only the seed parameter is unique for each DG, whereas the four other values are shared among the independent DGs. The insertion of triples via INSERT queries is not performed in equal time periods, but based on their real time stamp generation, emulating a realistic scenario. Additionally, during the benchmark execution, each DG will decrease the waiting time between triples insertions, in order to test the performance of the triple store when it is being overloaded with multiple requests in a short period of time.

2.2.3 Task Generator

The Task Generator (TG) job is two-fold:

- it sends the SELECT SPARQL queries to be performed against the triple store, and
- it sends the expected results of a particular SELECT query to the Evaluation Storage.

As we mentioned above, a SELECT query must be performed against the triple store, after a certain amount of INSERT queries are sent to the storage solution by one DG. The SELECT query is created by the DG, and along with the expected results, it is sent to the TG, who then sends the SELECT query to the triple store and the expected results to the Evaluation Storage. More details about the creation of the SELECT query and the expected results set can be found in Section 3. Please note that the current version of the benchmark creates one TG for all DGs that are present during the execution of the benchmark.

2.2.4 System Adapter

The System Adapter (SA) is the component that establishes the communication between the other benchmark components and the triple storage system. The SA component is independent of the benchmark and is initialized by the platform. The functionality of a SA is divided in 5 steps:

- Initialization of the storage system.
- Retrieval of triples in the form of an INSERT SPARQL query and insertion of the aforementioned triples into the storage by each DG.
- Retrieval of string representation of the graph name of each DG.
- Retrieval of SELECT SPARQL queries from the TG and execution of the queries against the storage system. The retrieved results are then sent to the Evaluation Storage component.
- Shut down of the storage system.

2.2.5 Evaluation Module

The Evaluation Module (EM) is responsible for evaluating:

- The overall performance of a triple store by deriving the micro and Macro-Average Recall, Precision and F-measure, the average triples per second, the maximum triples per second and the average answer time of the whole benchmark.
- The performance of each SELECT query in terms of Recall, Precision and F-measure, triples per seconds (if available) and answer time.

More information about ODIN’s KPIs can be found in Section 4.2.

2.3 Data Pre-processing and Streams

As described in Section 2.2, ODIN emulates a realistic scenario of data ingestion and retrieval. As a result, the insertion of triples into the storage solution must be performed based on the generated time stamp of each triple. To achieve such goal, every dataset that is received by each DG must follow a set of preprocessing steps during the initialization phase of a DG component, dividing the triples and transforming their original time stamps to fit the duration of the benchmark. First, each DG invokes a unique instance of the same mimicking algorithm, taking as parameters the DG’s seed value and the
population number. Once each dataset is received, the algorithm proceeds to the data preprocessing phase. The data preprocessing phase consists of the following consecutive steps:

1. Each DG divides the retrieved dataset into files based on the generation time stamp of each triple. Each file contains only triples that were generated at the same point in time.

2. Each DG sends the begin and end time stamp of their dataset to the BC.

3. Since the total duration of the benchmark is at most 20 minutes (including initialization of the components and evaluation of the system), we use a scaling down factor that converts the original time stamps into the time interval of the benchmark. Once all DGs have sent their begin and end time stamp to the BC, the BC sends back to all DGs the lowest begin time stamp ($OldMin$) and the highest end time stamp ($OldMax$) derived from the overall values of begin and end time stamps from all the DGs’ datasets. These two values serve as the overall begin and end point of the data that will be inserted into the storage solution.

4. The original time stamps of the triples are normalized to the benchmark’s interval using the following formula:

$$newTime\text{Stamp} = \frac{(originalTime\text{Stamp} - OldMin) \times (NewMax - NewMin)}{OldMax - OldMin} + NewMin$$

where $NewMax$ is the total duration of the benchmark and $NewMin$ is set to 0.

Once all the triples’ time stamps are normalized, each DG converts each data file into an INSERT SPARQL query. Then, given the number of INSERT queries after which a SELECT query will be performed, INSERT queries are divided into streams. Each stream consists of an equal number of INSERT queries, a SELECT query and the set of expected results for the SELECT query. Each query and expected result set are derived as follows:

- An INSERT query includes the set of triples that were generated at the same point in time. It also includes information about the graph that the corresponding DG will insert the queries. During the execution of the benchmark, an INSERT query will be sent to the SA after a particular delay calculated as the difference in time stamps between the previous INSERT query and its time stamp. Additionally, the delay between INSERT queries decreases among streams using the scaling factor $2^{streamID-1}$, where the $streamID \in [1, 2, ...]$ is the unique ID value of each stream. The delay between the INSERT queries of the last stream is set to 0.

- For each stream, a SELECT query is conducted by using the last INSERT query of the corresponding stream (Figure 1). Our goal is to check if all triples of the INSERT query have been inserted into the triple store. As a result, we create the minimum SELECT query in terms of length that covers all triples. Using the Least General Generalization (LGG) technique, we identify the all atomic triple patterns with one variable (Figure 2) and order the corresponding variables and results based on their frequency in triples (Table 1). We begin conducting the SELECT query by adding triple patterns that correspond to the most frequent variable-result pair (Figure 3). The procedure continues until all triples of the INSERT query are covered in the SELECT query. During the execution of the benchmark, each SELECT query is sent to the TG with a delay that is decreased among streams. For example, if the first SELECT query is sent $n$ milliseconds after the last INSERT query of the first stream, then the second SELECT query will be sent $\frac{n}{2^{streamID-1}}$ milliseconds after the last INSERT query of the second stream. The delay of the last SELECT query of the last stream is set to 0.
Once the INSERT and SELECT queries are conducted for a stream, the DG performs the INSERT queries against its own Jena TDB instance. Then, the SELECT query is performed and the resulting set is retrieved and stored as the expected results of the corresponding SELECT query (Table 2), in order to be sent to the evaluation storage during the benchmark execution.

Figure 2 illustrates the overall procedure of the pre-processing phase.

Listing 1: Example INSERT SPARQL query

```sparql
@prefix : <www.Odin.com#> .
INSERT DATA{
  :Event1 rdf:type :tweet .
  :Event1 :hasTP "2017-02-15" .
  :Event1 :hasMsg "Happy bday" .
  :Event2 :hasTP "2017-02-15" .
  :Event2 :hasMsg "18 again!" .}
```

Listing 2: Atomic triple patterns of example INSERT SPARQL query

```sparql
@prefix : <www.Odin.com#> .
(?x1 ?p1 :tweet
  :Event1
  :hasTP "2017-02-15"
  :hasMsg "Happy bday"
  :Event2
  :hasTP "2017-02-15"
  :hasMsg "18 again!"
...
```

Listing 3: Derived SELECT SPARQL query

```sparql
@prefix : <www.Odin.com#> .
SELECT ?x1 ?x2
WHERE{
  ?x1 rdf:type :tweet .
  ?x1 :hasTP "2017-02-15" .
  ?x1 :hasMsg "Happy bday" .
  ?x2 rdf:type :tweet .
  ?x2 :hasTP "2017-02-15" .
  ?x2 :hasMsg "18 again!" .}
```

3 Approach

Now that we have introduced each component individually and described the preprocessing data phase, we can proceed in explaining in detail the ODIN benchmark. The ODIN benchmark consists of five phases:

3.1 Initialization phase

- The BC initializes itself by loading the appropriate configuration from the HOBBIT platform GUI and initializes the set of DGs, the TG and the Evaluation Storage. Then it waits until it receives all begin and end dataset points from each DG. It computes the overall begin and end points of the mimicking datasets and sends it back to the DGs.

---

https://jena.apache.org/documentation/tdb/
Table 1: Frequency ordering of variable-result pairs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Result</th>
<th>Triple No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>?x1</td>
<td>:Event1</td>
<td>1 2 3</td>
</tr>
<tr>
<td>?x2</td>
<td>:Event2</td>
<td>4 5 6</td>
</tr>
<tr>
<td>?p1</td>
<td>rdf:type</td>
<td>1 4</td>
</tr>
<tr>
<td>?o1</td>
<td>:tweet</td>
<td>1 4</td>
</tr>
<tr>
<td>?p2</td>
<td>:hasTP</td>
<td>2 5</td>
</tr>
<tr>
<td>?o2</td>
<td>&quot;2017-02-15&quot;</td>
<td>2 5</td>
</tr>
<tr>
<td>?p3</td>
<td>:hasMsg</td>
<td>3 6</td>
</tr>
<tr>
<td>?o3</td>
<td>&quot;Happy bday&quot;</td>
<td>3</td>
</tr>
<tr>
<td>?o4</td>
<td>&quot;18 again!&quot;</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 2: Reference set of SELECT SPARQL query

<table>
<thead>
<tr>
<th>?x1</th>
<th>?x2</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Event1</td>
<td>:Event2</td>
</tr>
</tbody>
</table>

- The DGs receive the appropriate configuration values from the BC and proceed in completing step 1 and 2 of the Pre-processing phase. Once they receive the overall begin and end points from the BC, they continue executing the remaining steps of the Pre-processing phase before transforming the data into streams, as described above.

- The TG is initialized by the BC and waits for a signal to start.

- The SA is initialized by the platform. The initialization phase of each SA depends on the requirements of each triple store.

3.2 Loading phase

After the SA is running and the benchmark has started, the SA can receive data from the data queue, which must be loaded into the triple store. This can be done as bulk load.

- The DGs send UTF-8 encoded string representations of their graph URI to the SA and postpone their execution until a corresponding signal is received by the BC.

- The BC waits until each DG sends a message that it is finished with bulk load phase. Then, the BC sends a message to the SA that bulk load phase is over and waits until the SA sends a message back that it is also finished the bulk load phase. Once it receives the appropriate signal from the SA, it sends a message to all DGs, to proceed in executing their tasks.
3.3 Querying phase

- The BC sends a begin signal to the DGs and the TG and waits until all components have finished their execution.

- The DGs start sending INSERT queries to the SA to be performed against the triple store. At the end of each stream, the DG sends the corresponding SELECT query and its expected results to the TG.

- The TG receives the SELECT queries and the expected results from each DG. The SELECT queries are sent to the SA and the expected results are sent to the Evaluation Storage for evaluation of the task at the end of the benchmark. Note that the TG also sends additional information to the Evaluation Storage, such as the number of overall triples that are inserted during a stream, the begin point of the stream and the time point when the SELECT query was sent to the SA.

- The SA receives INSERT queries from the DGs and SELECT queries from the TG and performs them against the triple store. The results of the SELECT queries are sent to the evaluation storage along with the task unique identification number.

3.4 Evaluation Phase

- Once the DGs, the TG and the SA have finished their execution, the BC initializes the EM component and sends its signal to start the evaluation.
For each SELECT query, the EM receives the expected results, the received results, the time stamp that indicates when the SELECT query was sent to the SA and the time stamp that indicates when Evaluation Storage received results for that SELECT query. Then it proceeds in evaluating each task individually. Finally, it summarizes the evaluation of the benchmark using the KPIs described in Section 4.2.

3.5 Termination phase

Each component terminates its execution. Please note that the BC waits until every other component is terminated before it shuts down itself.

4 Evaluation

4.1 Baseline System: OpenLink Virtuoso

In this section, we are going to describe the OpenLink Virtuoso storage system that ODIN benchmarked as its baseline triple store. Our goal was to test the ability of Virtuoso OpenSource to insert and retrieve triples when presented with a real time scenario of triple ingestion. We evaluated OpenLink Virtuoso’s performance in terms of scalability and efficiency by conducting a set of experiments where we increased the number of triples, the number of streams and the number of DGs.

Virtuoso is a middleware and database engine hybrid that combines the functionality of a traditional Relational database management system (RDBMS), Object-relational database (ORDBMS), virtual database, RDF, XML, free-text, web application server and file server functionality in a single system. The reason behind our decision to use Virtuoso as a system to benchmark, is the ability of OpenLink to provide an open source edition of their system, dubbed OpenLink Virtuoso. As a result, in our following set of experiments, we used OpenLink Virtuoso and evaluated its performance of ingesting and retrieving triples.

In order for a system to be benchmarked by ODIN, it has to be able to interact with the Hobbit platform and the ODIN benchmark components. As a result, we have implemented a SA for OpenLink Virtuoso (VirtuosoSA) in the Java programming language. VirtuosoSA was created following the description of a SA in Section 2.2.

4.2 KPIs

Our evaluation consists of three KPIs:

- Recall, Precision and F-Measure: The INSERT queries created by each data generator will be send into a triple store by bulk load. After a stream of INSERT queries is performed against the triple store, a SELECT query will be conducted by the corresponding data generator. The SELECT query will be send to the TG along with its expected answers. Then, the TG will send the SELECT query to the SA and it will send the expected results in the evaluation storage. As explained above, once the SA performs the SELECT query against the triple store system, it

---

5 https://github.com/openlink/virtuoso-opensource
receives and sends the retrieved results to the evaluation storage. At the end of each experiment, we will compute the Recall, Precision and finally the F-measure of each SELECT query.

In Information Retrieval, Recall and Precision are used as relevance measurements and are defined in terms of retrieved results and relevant results for a single query. Recall is the fraction of relevant documents that were successfully retrieved.

\[
Recall = \frac{|\{\text{relevant results}\} \cap \{\text{retrieved results}\}|}{|\{\text{relevant results}\}|} \tag{2}
\]

and precision is the fraction of the retrieved documents that are relevant to a query.

\[
Precision = \frac{|\{\text{relevant results}\} \cap \{\text{retrieved results}\}|}{|\{\text{retrieved results}\}|} \tag{3}
\]

F-measure is the harmonic mean of Recall and Precision:

\[
F\text{-measure} = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}
\]

For our set of experiments, the relevant results for each SELECT query will be created prior to the system benchmarking by inserting and querying an instance of the Jena TDB storage solution. Note that each Data Generator will use its own graph and its own Jena TDB instance to insert data.

Additionally, we will compute:

\[
\text{Macro-Average-Precision} = \frac{\sum_{i=1}^{\lambda} Precision_i}{\lambda} \tag{5}
\]

\[
\text{Macro-Average-Recall} = \frac{\sum_{i=1}^{\lambda} Recall_i}{\lambda} \tag{6}
\]

\[
\text{Macro-Average-F-measure} = \frac{2 \times \text{Macro-Average Precision} \times \text{Macro-Average Recall}}{\text{Macro-Average Precision} + \text{Macro-Average Recall}} \tag{7}
\]

where \(\lambda\) is the number of SELECT queries performed against the storage solution during the execution of the benchmark and Micro and Macro-Average Recall, Precision and F-measure of the whole benchmark. The aforementioned measurements \(Precision_i\) and \(Recall_i\) are the precision and recall of the \(i\)-th SELECT query:

\[
\text{Micro-Average-Precision} = \frac{\sum_{i=1}^{\lambda} |\{\text{relevant results}_i\} \cap \{\text{retrieved results}_i\}|}{\sum_{i=1}^{\lambda} |\{\text{retrieved results}_i\}|} \tag{8}
\]

\[
\text{Micro-Average-Recall} = \frac{\sum_{i=1}^{\lambda} |\{\text{relevant results}_i\} \cap \{\text{retrieved results}_i\}|}{\sum_{i=1}^{\lambda} |\{\text{relevant results}_i\}|} \tag{9}
\]

\[
\text{Micro-Average-F-measure} = \frac{2 \times \text{Micro-Average Precision} \times \text{Micro-Average Recall}}{\text{Micro-Average Precision} + \text{Micro-Average Recall}} \tag{10}
\]

where the \(\{\text{relevant results}_i\}\) and \(\{\text{retrieved results}_i\}\) are the relevant and the retrieved results of the \(i\)-th SELECT query resp.

We have to mention that misclassifications between the expected and received results does not necessarily mean that OpenLink Virtuoso is prone to misclassify results or to have a bad performance, but that there are miss-matches for results sets between Jena TDB and OpenLink Virtuoso.
• Triples per second: at the end of each stream and once the corresponding SELECT query is performed against the system, we will measure the triples per second as a fraction of the total number of triples that were inserted during that stream. This is divided by the total time needed for those triples to be inserted (begin point of SELECT query - begin point of the first INSERT query of the stream). We will provide the maximum value of the triples per second of the whole benchmark. The maximum triples per second value is calculated as the triples per second value of the last stream with Recall value equal to 1.

• Average answer time: we will report the average answer delay between the time stamp that the SELECT query has been executed and the time stamp that the results are send to the evaluation storage. The first aforementioned time stamp is generated by the task generator and the second time stamp is generated by the evaluation storage. Additionally, we will provide the answer time of each SELECT query.

4.3 Results

We conducted a set of experiments to evaluate the performance of OpenLink Virtuoso. Throughout the experiments we used the same values for each of the following configurable parameters:

• Duration of the benchmark = 600,000 ms
• Name of mimicking algorithm algorithm output folder = output_data/
• Name of mimicking algorithm = TRANSPORT_DATA
• Seed for mimicking algorithm = 100
• Number of task generators - agents = 1

For the rest of the configurable parameters, Number of insert queries per stream, Population of generated data and Number of data generators - agents, we divided our experiments into three sets, in which we kept two of the aforementioned parameters stable and used a different set of values for the third.

The Number of data generators - agents influences the number of INSERT and SELECT queries that are send to the SA. By increasing the number of DGs, we scale out over the number of INSERT and SELECT queries generated by the benchmark.

The Population of generated data influences the size of the generated triples; increasing the amount of population indicates that the number of streams, INSERT and SELECT queries also increases. The size of each stream for this set of experiments will be the same since the Number of insert queries per stream has the same value for each experiment.

The Number of insert queries per stream influences the number of streams in which the data is divided; if the number of INSERT queries per stream increases, the number and size of streams and SELECT queries decreases.

4.3.1 Experiments with different numbers of data generators - agents

In this set of experiments we set:

• Number of insert queries per stream = 100
• **Population of generated data** = 1,000

and we conducted a set of experiments using 1, 2, 4, 8 and 16 DGs. By increasing the number of DGs, we scaled out over the amount of INSERT and SELECT queries that were sent to OpenLink Virtuoso. All experiments required at most 20 min overall runtime to be executed. During the experiments, OpenLink Virtuoso was able to retrieve results for all tasks.

![Figure 3](image-url)

**Figure 3**: Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1, 2, 4, 8 and 16.

Figure 3 illustrates the Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1, 2, 4, 8 and 16. We observe that on average, all the Micro and Macro-Average values of our evaluation measures decrease as the number of DGs increases. As a result, the performance of OpenLink Virtuoso decreases when we scale out over INSERT and SELECT queries. Additionally, the all the Micro-Average measures receive higher values compared to their corresponding Macro-Average values. Based on Equations 5, 6 and 7, Macro-Average measures give equal weights to all tasks and try not to be biased toward tasks with larger expected results. Whereas, based on Equations 8, 9 and 10, the Micro-Average measures tend to be biased towards highly populated result sets. As a result, this observation over Figure 3 indicates that tasks with smaller expected results are less correctly classified, compared to tasks with larger expected results.

Also, based on Figure 4, we notice that the average delay of tasks for Number of data generators - agents = 1, 2, 4, 8 is relatively low and similar among the different values of DGs. However when the system is introduced with 16 DGs, OpenLink Virtuoso faces a higher amount of SELECT queries and the time between sending the query and processing it increases significantly.

By observing Figure 5, we see that the highest Triples-Per-Second value is obtained when the system is introduced with 2 DGs. For larger values of DGs, the maximum number of Triple-Per-Second decreases significantly. Additionally we have to mention that for DG = 1 and 2, the maximum Triples-Per-Second value is obtained by the final task, which indicates that the recall of the last task is 1. Whereas, for DG = 4, 8 and 16, the last task that receives recall = 1 is the 38th (out of 41 tasks), the 26th (out of 82 tasks) and 30th (out of 164 tasks) respectively. As a result, the miss-matches between OpenLink Virtuoso and Jena TDB increase when the system has to process a larger amount of INSERT and SELECT queries.

Figures 6, 7, 8, 9 and 10 illustrate a detailed description of the precision, recall and f-measure of each individual task for the different values of the DGs. We see that precision (and as a result f-measure) receives lower values on average as the number of DGs increases, compared to recall.
Figure 4: Average Delay of tasks for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1, 2, 4, 8 and 16.

Figure 5: Maximum Triples-per-Second for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1, 2, 4, 8 and 16. The result for 16 DG is not visible in the figure because it is very low compared to the other values (38.65).

Observation indicates that once OpenLink Virtuoso is introduced with a higher amount of queries, it requires more time to process them and as a result, the received results include a small or no amount of the expected results. Also, as we mentioned previously, for 16 DGs, the system performance decreases significantly, since after the 26th task all measure receive 0 values.

Additionally, for the first set of experiments we provide the delay between the time point that a SELECT query was sent to the system and the time point that a response has been received from the system. For DG = 1 and 2, we observe that the task delays increase over time (Figures 11, 12), and OpenLink Virtuoso achieves the highest delay in the final task. For DG = 4 and 8 (Figures 13, 14) where the number of tasks increases, we observe that the function of task delays over time has one and two peaks resp. This indicates that OpenLink Virtuoso becomes overloaded with tasks at one
and two time points resp. and once it processes the queries that were waiting in the queue, the task delays decreases. However, we also notice the system’s attitude for DG = 16 (Figure 15) follows the OpenLink Virtuoso performance for DG = 1 and 2.

### 4.3.2 Experiments with different population values of generated data

In this set of experiments we set:

- **Number of insert queries per stream** = 100
- **Number of data generators - agents** = 4

and we conducted a set of experiments using a population of 1,000, 5,000, 10,000 and 25,000
Figure 8: Recall, Precision and F-measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 4. The horizontal axis represents the number of tasks.

Figure 9: Recall, Precision and F-measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 8. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...30. The remaining tasks had 0 in all three measurements.

generated data. By increasing the number of generated data, we scale out over the number of streams and as a result the number of INSERT and SELECT queries.

Figure 16 illustrates the Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 1,000, 5,000, 10,000 and 25,000. We observe that as the population of generated data increases, OpenLink Virtuoso classification performance decreases significantly. Even when we introduced the system with only 5 times higher population, Micro-Average-Recall and F-measure, and all the Macro-Average measures dropped by 21% on average. However, we notice that although Micro-Average-Precision receives decreases, it achieves significantly higher values compared to the other Micro and Macro-Average values. Based on Equation 8, Micro-Average-Precision is influenced only by tasks that are able to retrieve a non-zero result set. Whereas, the Macro-Average Precision that is calculated as an average of precision among tasks is highly influenced by tasks where OpenLink Virtuoso received
Figure 10: Recall, Precision and F-measure for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 16. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1…40. The remaining tasks had 0 in all three measurements.

Figure 11: Task Delay for Number of insert queries per stream = 100, Population of generated data = 1,000 and Number of data generators - agents = 1. The horizontal axis represents the number of tasks.

mostly irrelevant or empty results.

As expected, based on Figure 26, we notice that the average delay of tasks for Population of generated data = 1,000, 5,000, 10,000 and 25,000 increases as OpenLink Virtuoso is introduced with more generated data.

Furthermore, by observing Figure 18, we see that the highest Triples-Per-Second value is obtained when the system is introduced with Population of generated data = 1,000. For larger values of generated data, the maximum number of Triple-Per-Second decreases significantly. Additionally we have to mention that for Population of generated data = 1,000, the maximum Triples-Per-Second value is obtained by the 38th task (out of 41 tasks). Whereas, for the remaining population values, the last task that receives recall = 1 is the 55th (out of 200 tasks), the 104th (out of 395 tasks) and 149th (out of 968 tasks) respectively (Figures 19, 20 and 21). As a result, the miss-matches between
OpenLink Virtuoso and Jena TDB increase when the system has to process a larger amount of INSERT and SELECT queries.

Figures 8, 19, 20 and 21 illustrate a detailed description of the precision, recall and f-measure of each individual task for the different values of the DGs. We come to a similar conclusion as in the first set of experiments, that once OpenLink Virtuoso is introduced with a higher number of streams, INSERT and SELECT queries, precision (and as a result f-measure) receives lower values on average.
Additionally, for **Population of generated data** = 5,000, 10,000 and 25,000, on average 77% of the tasks receive 0 values for all three measurements.

Additionally, as in the first set of experiments, we provide the delay between the time point that a SELECT query was sent to the system and the time point that a response has been received from the system. For all different values of generated data, we observe that the task delay increases monotonically with the size of the tasks (Figures 13, 22, 23 and 24).

### 4.3.3 Experiments with different numbers of insert queries per stream

In this set of experiments we set:

---

**Figure 14**: Task Delay for **Number of insert queries per stream** = 100, **Population of generated data** = 1,000 and **Number of data generators - agents** = 8. The horizontal axis represents the number of tasks.

**Figure 15**: Task Delay for **Number of insert queries per stream** = 100, **Population of generated data** = 1000 and **Number of data generators - agents** = 16. The horizontal axis represents the number of tasks.
Figure 16: Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 1,000, 5,000, 10,000 and 25,000.

Figure 17: Average Delay of tasks for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 1,000, 5,000, 10,000 and 25,000.

- Population of generated data = 10,000
- Number of data generators - agents = 4

and we conducted a set of experiments using 100,500,1,000 and 5,000 INSERT queries after which a SELECT query will be performed. By decreasing the number of INSERT queries per streams, the generated data is divided into more streams and as a result, we scale out over the number and size of streams and SELECT queries.

Figure 25 illustrates the Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 100, 500, 1,000 and 5,000. Apart from Macro-Average Precision, all other measures receive noticable low values for the values values of INSERT queries. Note that, Macro-Average
Figure 18: Maximum Triples-per-Second for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 1,000, 5,000, 10,000 and 25,000.

Figure 19: Recall, Precision and F-measure for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 5,000. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...70. The remaining tasks had 0 in all three measurements.

Precision behaviour has been explained previously for the second set of experiments. Additionally, we observe that as the number of INSERT queries increases, OpenLink Virtuoso classification performance increases on average.

By observing Figure 26, we notice that in most cases, the average delay of tasks is not influenced by the different numbers of INSERT queries per stream and is relatively slow. However, for Number of insert queries per stream = 500 we observe an outlier in the performance of OpenLink Virtuoso. Additionally, based on Figure 27, the maximum number of Triples-Per-Second is not influenced as well by the number of streams, since all different configurations receive similar results.

Figures 20, 28, 29 and 30 illustrate a detailed description of the precision, recall and f-measure of each individual task for the different number of INSERT queries per stream. For Number of insert
Figure 20: Recall, Precision and F-measure for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 10,000. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...110. The remaining tasks had 0 in all three measurements.

Figure 21: Recall, Precision and F-measure for Number of insert queries per stream = 100, Number of data generators - agents = 4 and Population of generated data = 25,000. The horizontal axis represents the number of tasks. Please note that the graph includes Recall, Precision and F-measure values for tasks 1...160. The remaining tasks had 0 in all three measurements.

queries per stream = 500, 1,000 and 5,000, on average after the 6th task, all measurements receive 0 values. As a result, the performance of OpenLink Virtuoso in individual tasks is not influenced by the number of streams per experiment.

Figures 23, 31, 32 and 33 present the task delay of each task for the different values of INSERT queries per stream. For Number of insert queries per stream = 100 and 500, we observe that the task delays increase monotonically over time, whereas for Number of insert queries per stream = 1,000 and 5,000, the function of task delay shows local minima and maxima. As expected, the overall delays however decrease when the number of streams increases.

4.3.4 Primarily results

For our final experiment, we present the primarily results of benchmarking OpenLink Virtuoso, ONTOS Quad developed by Ontos and Virtuoso Commercial 8.0. This experiment was conducted as a part
of the Mighty Storage Challenge (MOCHA2017) \(^8\) that was accepted and presented in the Extended Semantic Web Conference (ESWC) in 2017. ODIN participated in Task 1 as the benchmark whose goal was to evaluate the performance of storage solutions regarding their ability to ingest RDF data.

All systems were given 25 mins to complete the insertion and retrieval of triples and the overall benchmark did not exceed 40 mins. In the following figures, the instance of OpenLink Virtuoso that was used for the previous set of experiments is referred to as Baseline for Task 1. Additionally, OpenLink Virtuoso served as the baseline system for all MOCHA2017 tasks, and we were provided with with a SA for OpenLink Virtuoso that was able to participate in all tasks (MOCHA Baseline). One of the main differences between Baseline for Task 1 and MOCHA Baseline is the way that OpenLink Virtuoso communicates with the SA. For Baseline for Task 1, the SA creates within its docker container another container for OpenLink Virtuoso, whereas MOCHA Baseline invokes a script that runs an instance of

---

\(^8\)https://project-hobbit.eu/challenges/mighty-storage-challenge/
Figure 24: Task Delay for **Number of insert queries per stream** = 100, **Number of data generators - agents** = 4 and **Population of generated data** = 25,000. The horizontal axis represents the number of tasks.

Figure 25: Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure for **Population of generated data** = 10,000, **Number of data generators - agents** = 4 and **Number of insert queries per stream** = 100, 500, 1,000 and 5,000.

Each MOCHA2017 task’s organizer must announce a winner for their task and the based on the majority vote, the system with the most wins would be announced as the winner of the challenge. However, the performance of the systems for the other MOCHA2017 tasks is beyond the scope of this review and we will focus on the performance of each system after being benchmarked by ODIN.

For this experiment, all parameters were set to their default values, except from the following:

- **Population of generated data** = 10,000
- **Number of data generators - agents** = 4 and
- **Number of insert queries per stream** = 100

The total number of SELECT queries that each system had to process was 395.
By observing Figure 34, we notice that *Virtuoso Commercial 8.0* has by far the best performance compared to the other two systems in terms of Macro and Micro-Average Precision, Recall and F-measure. *Virtuoso Commercial 8.0* was able to store and retrieve more triples through out the whole benchmark. However, the maximum performance value was achieved for Micro-Average Recall = 0.67, which indicates that the miss classifications between the Jena TDB and *Virtuoso Commercial 8.0* were still high on average. Additionally, since the Micro-Average values were higher compared to the Macro-Average values, we can conclude by stating that *Virtuoso Commercial 8.0* was able to retrieve more relevant triples to a SELECT query, for tasks with higher quantity of expected results.

Furthermore, we also notice that the Micro-Average Precision of *ONTOS Quad* is higher that the other systems. As explained previously, the Micro-Average values are calculated only when there are non-zero received results for a task. *ONTOS Quad* was able to retrieve results for the first 7 tasks and for the remaining 388 tasks, the system returned 0 received results or included 0 relevant results in its...
received result set as shown in Figure 36. Overall, by observing Figures 37 and 36, we notice that Virtuoso Commercial 8.0 is the only system that was able to retrieve non-zero results for the majority of the SELECT queries.

In terms of maximum Triples-per-Second, based on Figure 38, we notice that Virtuoso Commercial 8.0 was able to achieve the highest maximum TPS at the latest task possible. It receives the last recall value of 1 at task 358 (out of 395), whereas the other systems have issues with recall at much earlier stages of the benchmark. Especially for the ONTOS Quad system, we see that its recall drops significantly after the 6th SELECT query.

Also, we need to mention that ONTOS Quad and Virtuoso Commercial 8.0 were not able to perform all select queries within 25 mins. ONTOS Quad was not able to send results to the evaluation...
storage throughout the whole benchmark, whereas Virtuoso Commercial 8.0 was not able to execute SELECT queries after 358 tasks, which is one of the reasons why its recall drops to 0.

Additionally, we present the task delay for each task for all systems in Figure 39. We notice that all systems have a relatively low task delay over the set of SELECT queries. Whereas Virtuoso Commercial 8.0 has a monotonically ascending task delay function, that drops to 0 after the 358th task, since the system is no longer available because it exceeded the maximum allowed time to process queries.

Finally, we notice that even though Baseline for Task 1 and MOCHA Baseline used the same triple store system, they have difference regarding the maximum TPS and the task delay over tasks. This can be partially justified by the fact that those two SAs use a different way to communicate with
Finally, the winner of Task 1 of the MOCHA2017 challenge was *Virtuoso Commercial 8.0* that had a better overall performance in terms of data ingestion and retrieval.

Figure 32: Task Delay for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 1,000. The horizontal axis represents the number of tasks.

Figure 33: Task Delay for Population of generated data = 10,000, Number of data generators - agents = 4 and Number of insert queries per stream = 5,000. The horizontal axis represents the number of tasks.
Figure 34: Micro-Average-Recall, Micro-Average-Precision, Micro-Average-F-Measure, Macro-Average-Recall, Macro-Average-Precision, Macro-Average-F-Measure of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline. Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.

Figure 35: Average Delay of tasks of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline. Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.
Figure 36: Precision of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline. Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.

Figure 37: Recall of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline. Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.
Figure 38: Maximum Triples-per-Second of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline. Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.

Figure 39: Task Delay of Baseline for Task 1, ONTOS Quad, Virtuoso Commercial 8.0 and MOCHA Baseline. Number of insert queries per stream = 100, Population of generated data = 10,000 and Number of data generators - agents = 4.
5 Conclusion and Future Work

In this report we presented the first version of ODIN benchmark for storage and retrieval of streamed data. The aim of ODIN is to benchmark the performance of triple store solutions when faced with streams of data from industrial machinery in terms of efficiency and completeness. ODIN’s goal is to evaluate triple storage solutions, focusing on data volume and velocity.

As part of our report, we gave an overview of the system and its configurable parameters as they are displayed in the HOBBIT platform. We described in detail the components of ODIN and their role in the benchmark process, and we also presented the pre-processing phase of the data and how it is transformed into streams. Then, we gave a detailed overview of our approach for evaluating triple stores, by describing each phase of the benchmark. Also, we conducted a set of experiments using a baseline storage system and we presented our evaluation and results, after describing the KPIs of our evaluation. Finally, we presented a set of results as part of Task 1 of the MOCHA2017 challenge that was organized for ESWC 2017.

Our primary results from Section 4.3 show that ODIN is a well structured and promising benchmark for evaluating triple storage solutions that can process INSERT and SELECT SPARQL queries. Our goal was to develop a realistic scenario of triple insertion and selection. For example, ODIN’s emulation can be compared to a predictive maintenance framework that checks the health state of 100 machines and sends complex queries to check for faulty situations that require immediate solutions. Based on our evaluation, the number of DGs and the population of triples highly influence the performance of a triple store in terms of Precision, Recall, F-measure and task delay. The performance of a triple store decreases in the presence of more independent agents that try to insert and retrieve triples in parallel.

As part of our future work for the second version of ODIN, we will:

- increase our evaluation scope by benchmarking more triple storage solutions;
- explore alternative approaches for constructing SELECT queries;
- tweak the Duration of the benchmark parameter of the benchmark to observe how the performance of a triple store is influenced;
- increase the volume and velocity of the data and the DGs further;
- incorporate alternative triple store solutions for retrieving a reference set for a SELECT query.