

**Developing  
Applications  
Enhancements**

**Data-Intensive  
with Iterative**

**Cloud  
Quality**



# **DICE Testing tools – Final version**

**Deliverable 5.5**

---

|                             |   |
|-----------------------------|---|
| <b>Deliverable:</b>         | D5.5  |
| <b>Title:</b>               | DICE Testing Tools – Final version  |
| <b>Editor(s):</b>           | Giuliano Casale (IMP)   |
| <b>Contributor(s):</b>      | Tatiana Ustinova (IMP), Gabriel Iuhasz (IEAT), Andrew Phee (FLEXI)                          |
| <b>Reviewers:</b>           | Matej Artač (XLAB), Ioan Dragon (IEAT)  |
| <b>Type (R/DEM/DEC):</b>    | Demonstrator  |
| <b>Version:</b>             | 1.0   |
| <b>Date:</b>                | 27-July-2017  |
| <b>Status:</b>              | Final version   |
| <b>Dissemination level:</b> | Public  |
| <b>Download page:</b>       | <a href="http://www.dice-h2020.eu/deliverables/">http://www.dice-h2020.eu/deliverables/</a> |
| <b>Copyright:</b>           | Copyright © 2017, DICE consortium – All rights reserved                                     |

---

#### DICE partners

---

|               |  |
|---------------|--|
| <b>ATC:</b>   | Athens Technology Centre                           |
| <b>FLEXI:</b> | Flexiant Limited                                   |
| <b>IEAT:</b>  | Institutul E Austria Timisoara                     |
| <b>IMP:</b>   | Imperial College of Science, Technology & Medicine |
| <b>NETF:</b>  | Netfective Technology SA                           |
| <b>PMI:</b>   | Politecnico di Milano                              |
| <b>PRO:</b>   | Prodevelop SL                                      |
| <b>XLAB:</b>  | XLAB razvoj programske opreme in svetovanje d.o.o. |
| <b>ZAR:</b>   | Universidad De Zaragoza                            |

---



The DICE project (February 2015-January 2018) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644869

## Executive summary

In this deliverable we present the final releases of the DICE Quality Testing (QT) tool and the DICE Fault Injection (FIT) tool. In its final version, QT allows to load test streaming processing applications developed using either Storm or Kafka, and consequently also other frameworks such as Spark Streaming that can acquire stream data from a Kafka data pipeline. QT allows to reproduce empirical data from a log trace of streaming messages and generate statistically similar traces. Moreover, it offers the ability to scale the arrival rates and volumes of data sent to the application in order to analyze its response under increasing data volumes and velocity. We also present the *DMON-gen* utility allows the developer to test how the DIA will behave when using different platform (i.e. Yarn, Spark, etc.) settings and use the resulting monitoring data to create training and validation sets that are later used by the anomaly detection tool. Lastly, we discuss the final version of the FIT tool, which allows to simulate faults in the infrastructure used by the DIA to assess its resiliency.

## Glossary

|          |   |
|----------|---|
| DIA      | Data-Intensive Application  |
| DICE     | Data-Intensive Cloud Applications with iterative quality enhancements |
| DMON     | DICE Monitoring Platform  |
| MMAAP    | Marked Markovian Arrival Process                                      |
| QoS      | Quality of Service  |
| QT       | Quality Testing   |
| QT-GEN   | QT workload generator   |
| QT-LIB   | QT library  |
| UML      | Unified Modelling Language  |
| YARN     | Apache Yet Another Resource Negotiator                                |
| API      | Application Programming Interface                                     |
| FIT      | Fault Injection Tool  |
| JSON     | JavaScript Object Notation  |
| VM       | Virtual machine   |
| CPU      | Central processing unit   |
| UI       | User interface  |
| GUI      | Graphical user interface  |
| ADT      | Anomaly detection tool  |
| DMON-GEN | DMON anomaly generator  |

## Table of contents

|   |    |
|---|----|
| Executive summary .....   | 3  |
| Glossary .....  | 4  |
| Table of contents .....   | 5  |
| 1 Introduction .....  | 7  |
| 1.1 Research and Technical Achievements in Year 3.....                | 7  |
| Achievement 1: QT-LIB support for Apache Kafka .....                  | 7  |
| Achievement 2: QT-LIB integration with DMON monitoring platform ..... | 7  |
| Achievement 3: workload testing for anomaly detection .....           | 7  |
| Achievement 4: fault-injection tool front-end .....                   | 7  |
| 1.2 Requirements .....  | 7  |
| 1.3 ‘Must have’ requirements .....                                    | 8  |
| 1.4 ‘Should have’ requirements .....                                  | 9  |
| 1.5 QT, FIT and DICE Architecture .....                               | 10 |
| 1.6 Deliverable organisation .....                                    | 10 |
| 2 Quality Testing (QT) .....  | 11 |
| 2.1 QT-LIB extension: an overview of Apache Kafka .....               | 11 |
| 2.2 QT-LIB for Apache Kafka.....                                      | 11 |
| 2.2.1 Baseline: kafka-perf-tool .....                                 | 11 |
| 2.2.2 Customization and extension.....                                | 11 |
| 2.2.3 Final QT-LIB internal architecture .....                        | 12 |
| 2.2.4 Using QT-LIB for Kafka .....                                    | 13 |
| 2.2.5 Kafka support validation .....                                  | 14 |
| 2.3 Extended integration with Storm and DMON .....                    | 15 |
| 2.3.1 DMON integration.....   | 15 |
| 2.3.2 Storm integration .....   | 16 |
| 3 Fault Injection Tool (FIT).....                                     | 18 |
| 3.1 Overview .....  | 18 |
| 3.2 Motivation.....   | 18 |
| 3.3 Design .....  | 18 |
| 3.4 Operation.....  | 19 |
| 3.5 Graphical User Interface .....                                    | 19 |
| 3.1 Integration .....   | 21 |
| 3.2 Validation.....   | 22 |

|       |                                       |    |
|-------|---------------------------------------|----|
| 3.3   | Installation.....                     | 23 |
| 4     | DMON-Gen Utility.....                 | 24 |
| 4.1   | General Overview .....                | 24 |
| 4.2   | Architecture.....                     | 24 |
| 4.3   | Monitoring data generation.....       | 25 |
| 4.4   | Use-cases and configuration .....     | 26 |
| 4.5   | Validation.....                       | 27 |
| 5     | Conclusion .....                      | 30 |
| 5.1   | Summary of achievements .....         | 30 |
|       | References .....                      | 31 |
| 6     | Annex.....                            | 32 |
| 6.1   | Parameter settings for DMON-gen ..... | 32 |
| 6.1.1 | Parameters for HDFS.....              | 32 |
| 6.2   | Parameters for Yarn service.....      | 36 |
| 6.3   | Parameters for Spark service .....    | 42 |

## 1 Introduction

In this final version of the Quality Testing (QT) tools, we discuss the year 3 achievements in relation to testing of data intensive application (DIA). Compared to the initial version of this deliverable (D5.4, released at M24), D5.5 covers the final versions of the load testing tools (QT-GEN, QT-LIB) and the fault injection tool (FIT).

### 1.1 Research and Technical Achievements in Year 3

In D5.4, we described two tools, QT-GEN and QT-LIB, for load testing Storm-based applications. The decision of focussing on this technology platform was motivated by the observation that other DICE technologies such as Apache Hadoop or Cassandra can be stress tested with mature open source tools like Apache JMeter. At the end of D5.4 we set a goal of extending the capability of QT to other DICE technologies like Apache Spark streaming. Moreover, in D5.6 we presented an initial release of FIT, which we now complete with this deliverable adding in particular a GUI.

#### *Achievement 1: QT-LIB support for Apache Kafka*

After reviewing the literature, we concluded that if QT could be extended to support Apache Kafka, then the tool would be in condition to load test a variety of stream processing platforms, including Apache Spark streaming, which can natively consume from Apache Kafka itself.

#### *Achievement 2: QT-LIB integration with DMON monitoring platform*

The quality testing API exposed by QT-LIB has been extended to support direct calls to DMON monitoring. In the case of Storm technology, QT-LIB can also directly retrieve performance data from Storm's internal monitoring system. In year 3 we have also defined a template for the end-user to automate the control of load testing experiments using monitoring data. This has been applied against the NewsAsset case study developed by ATC.

#### *Achievement 3: workload testing for anomaly detection*

In year 3 we have also developed testing tools that support DICE engineers in training the anomaly detection algorithms offered with the DMON monitoring platform. The tool, called *DMON-gen*, allows to repeatedly run the application using multiple configurations and workloads.

#### *Achievement 4: fault-injection tool front-end*

Lastly, we present the final version of the fault injection tool (FIT), which now supplies a graphical user interface that simplifies its use from within the DICE IDE environment.

### 1.2 Requirements

Deliverable *D1.2 - Requirement specification* [1] presents the requirement analysis for the whole project, including the QT tool. This section provides an updated list of requirements for QT and FIT at month 30. The actors are as follows:

- QTESTING\_TOOLS: the DICE Quality Testing tool
- CI\_TOOLS: the DICE Continuous Integration tools
- QA\_TESTER: the developer or operator interested in validating the application quality.

The main changes concerning the earlier version of the requirements is that we have revisited them to be consistent with the technology platforms that are supported by QT, namely stream-processing

systems. In particular the earlier requirements R15.5.x related to safety and anomaly detection are now under the remit of the anomaly detection and trace checking tools and obsolete for QT.

### 1.3 ‘Must have’ requirements

|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.6   |
| <b>Title</b>        | Test workload generation   |
| <b>Priority</b>     | Must have  |
| <b>Description:</b> | The QTESTING_TOOLS MUST be able to generate the workload with pre-specified characteristics for the APPLICATION. |

|                     |   |
|---------------------|---|
| <b>ID</b>           | R5.8.2  |
| <b>Title</b>        | Starting the quality testing  |
| <b>Priority</b>     | Must have   |
| <b>Description:</b> | The QTESTING_TOOLS MAY be invoked by the CI TOOLS or by the QA_TESTER |

|                     |   |
|---------------------|---|
| <b>ID</b>           | R5.8.3  |
| <b>Title</b>        | Test run independence   |
| <b>Priority</b>     | Must have   |
| <b>Description:</b> | The QTESTING_TOOLS MUST ensure that no side effects from past or ongoing tests leak into the runtime of any other test. |

|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.8.5   |
| <b>Title</b>        | Test outcome   |
| <b>Priority</b>     | Must have  |
| <b>Description:</b> | The QTESTING_TOOLS MUST provide the test outcome to CI_TOOLS: success or failure |

|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.13  |
| <b>Title</b>        | Test the application for efficiency  |
| <b>Priority</b>     | Must have  |
| <b>Description:</b> | The QTESTING_TOOLS MUST be capable of running tests with any configuration provided to it. |

|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.14.1  |
| <b>Title</b>        | Test the behaviour when resources become exhausted   |
| <b>Priority</b>     | Must have  |
| <b>Description:</b> | The QTESTING_TOOLS MUST provide the ability to saturate and exhaust resources used by the application. |



|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.17  |
| <b>Title</b>        | Quick testing vs comprehensive testing   |
| <b>Priority</b>     | Must have  |
| <b>Description:</b> | The QTESTING_TOOLS MUST receive as input parameter the scope of the tests to be run. |

|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.3.1   |
| <b>Title</b>        | VM Fault deployment  |
| <b>Priority</b>     | Must have  |
| <b>Description:</b> | The Fault Injection Tool MUST be able to cause faults on Virtual Machines. |

#### 1.4 ‘Should have’ requirements

|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.7   |
| <b>Title</b>        | Data loading support   |
| <b>Priority</b>     | Should have  |
| <b>Description:</b> | DEPLOYMENT_TOOLS and QTESTING_TOOLS SHOULD support bulk loading and bulk unloading of the data for the core building blocks. |

|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.14.2  |
| <b>Title</b>        | Trigger deliberate outages and problems to assess the application’s behaviour under faults                         |
| <b>Priority</b>     | Should have  |
| <b>Description:</b> | The QTESTING_TOOLS SHOULD use the fault injection environments functionality to test the application's resilience. |

|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.7.2   |
| <b>Title</b>        | Data feed actuator   |
| <b>Priority</b>     | Should have  |
| <b>Description:</b> | QTESTING_TOOLS SHOULD provide an actuator for sending generated or user-provided data to the application under test. |

|                     |   |
|---------------------|---|
| <b>ID</b>           | R5.3.2  |
| <b>Title</b>        | Integration with DICE deployment service  |
| <b>Priority</b>     | Should have   |
| <b>Description:</b> | The Fault Injection Tool SHOULD interface with the DICE deployment service in order to cause faults on various VMs used for a deployment. |

|                     |  |
|---------------------|--|
| <b>ID</b>           | R5.3.3   |
| <b>Title</b>        | FIT GUI  |
| <b>Priority</b>     | Should have  |
| <b>Description:</b> | The Fault Injection Tool SHOULD have a graphical user interface. |

## 1.5 QT, FIT and DICE Architecture

The Quality Testing and Fault Injection tools in the context of the DICE architecture are highlighted in Figure 1:

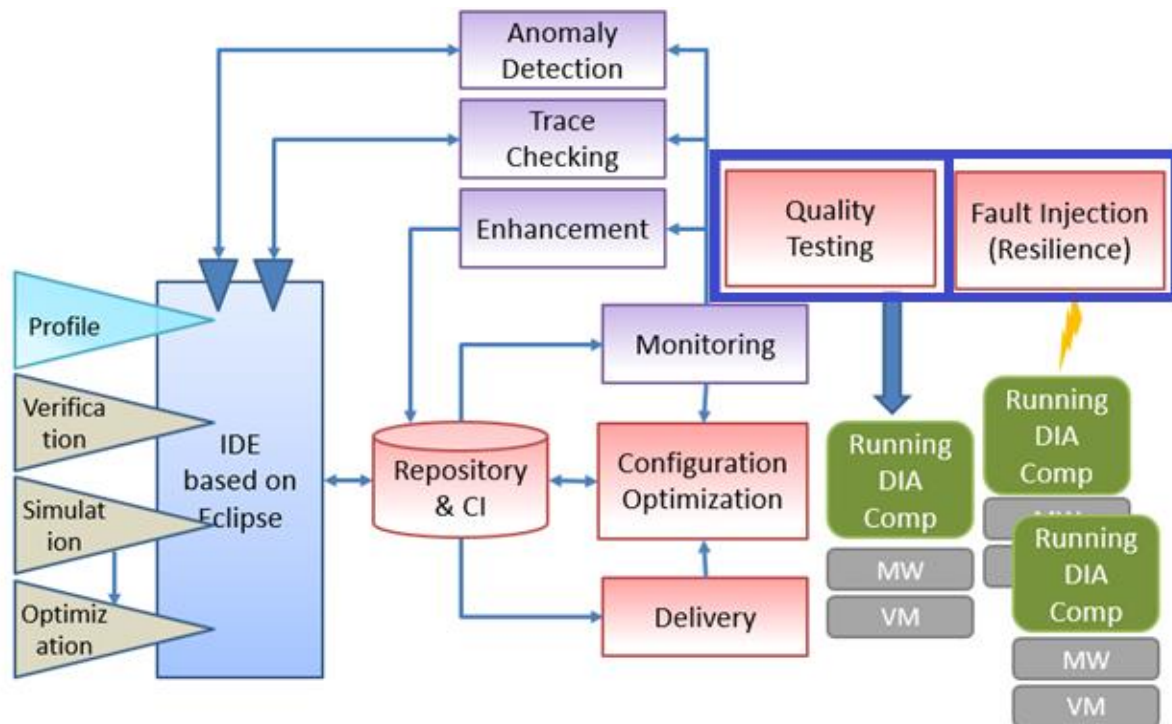


Figure 1 DICE Architecture - QT and FIT tools highlight

As shown in the figure, QT is a stand-alone component, which is aimed at injecting load into specific running components of the data-intensive application. QT tools can be used together with the fault injection tool (FIT) to assess the quality of the application in the presence of specific faults, such as simulated interruption of service for one or more VMs, or saturated resources that compromise performance up to reaching unavailability of one or more services. Both tools are meant to assess the quality of a Big data application, but QT has an emphasis on load-testing, whereas FIT focuses on checking the resilience to faults.

## 1.6 Deliverable organisation

The rest of this deliverable is organised as follows.

- In Chapter 3 we provide an overview of Apache Kafka and present the extended QT tool.
- In Chapter 4 we describe the enhancements to the Fault Injection tool delivered in year 3.
- In Chapter 5 we introduce the *DMON-gen* utility and how it is used to generate anomalies.
- In Chapter 6, we give conclusions assessing requirement fulfillment.
- The final Appendix gives additional material related to the FIT.

Source code, installation instructions and documentation of the tools can be found at:

<https://github.com/dice-project/DICE-Knowledge-Repository/wiki/DICE-Knowledge-Repository#quality>

<https://github.com/dice-project/DICE-Knowledge-Repository/wiki/DICE-Knowledge-Repository#fault>

## 2 Quality Testing (QT)

In the next sections, we introduce the main extensions developed for QT in Year 3. We begin by reviewing Apache Kafka support, and subsequently we discuss improved support for DMON and Storm.

### 2.1 QT-LIB extension: an overview of Apache Kafka

Apache Kafka is a multi-purpose distributed streaming platforms, which can be used to build real-time processing applications. In the context of DICE, Kafka is primarily supported to define high-throughput data pipelines that can be used to ingest high-velocity high-volume data streams into streaming processing systems such as Spark or Storm.

The general architecture of Kafka is as follows. Kafka supplies streams of *records*, grouped into *topics*. For each topic, Kafka maintains a set of partitions, where records are appended in sequential order. Partitions are used to distribute the records for a same topic across different servers and in this way enable scalable processing and load-balancing. Each topic has one or more subscribers that are automatically notified when a new record becomes available. A customizable retention period also allows to recover records that have been already pushed to subscribers. It is important to note that Kafka data structures are meant for high-performance access to streams.

Four APIs are made available to end users: *Production API* and *Consumer API*, which allow to publish and subscribe topics on Kafka; *Streams API*, which allow to process records in transit on one of more topic, sending the results to output topics; *Connector API*, which are used for integrating Kafka with existing applications and data sources (e.g., databases). The QT-LIB extension makes use primarily of the *Production API*.

### 2.2 QT-LIB for Apache Kafka

#### 2.2.1 Baseline: kafka-perf-tool

In order to extend the QT-LIB load testing tool to Apache Kafka, we have customized, extended, and integrated in QT-LIB an existing Java codebase for Kafka performance testing, which is the one underpinning the *kafka-perf-tool*<sup>1</sup>.

*kafka-perf-tool* is an open source tool (Apache licensed) for load testing, which provides a customizable JSON-based interface to specify the characteristics of the workload sent to a Kafka instance. *kafka-perf-tool* is very flexible, as it allows to parallelize production and consumption of Kafka topics and records, either generated at random or consumed from a static file. Among the main customizable parameters offered natively by *kafka-perf-tool* we find:

- Control over the duration and concurrency level of a test experiment
- Control over the number of producers, on the volume of messages they send and the message sizes, configurable strategies to allocate data to topics and partitions therein.
- Control over the number of consumers, on the topics to receive records from, and the preferred polling intervals and number of messages to receive.

As these were the features we were seeking to add to QT, we have decided to develop the QT-LIB extension starting from this baseline.

#### 2.2.2 Customization and extension

Within DICE, we integrate and extend the capabilities of *kafka-perf-tool* to address some limitations that restrict its use within the DICE methodology.

---

<sup>1</sup> <https://github.com/jkorab/ameliant-tools/tree/master/kafka/kafka-perf-tool>

The main limitations we identified were as follows:

- **Baseline limitation 1:** In its official distribution, when *kafka-perf-tool* was configured to issue pre-generated messages to the Kafka system, the tool was repeatedly publishing the same message multiple times, which is appropriate for benchmarking purposes but insufficient for reproducing an actual workload. In a realistic workload one would expect to reproduce data with varying messages types and message intensities.

**QT-LIB Solution:** QT-LIB includes a customized version of *kafka-perf-tool* that publishes different messages, in the order in which they are read from an external file trace. Such trace is generated in output by QT-GEN, similar to what done in the case of Storm technology. Our extension allows the user to *cyclically* read data from such an external trace file, so that the number of messages generated can be longer than the original trace itself. This is particularly useful to generate realistic workloads that would be expensive to acquire (e.g., new Twitter traces).

- **Baseline limitation 2:** Another limitation of *kafka-perf-tool* is that it does not allow to use a custom release time for each message. The user can only specify the number of messages and the duration of the testing experiment. This was perceived as a limitation compared to what QT-LIB offers for the Storm environment, where the user is allowed to replay the exact time series of records as they were originally exposes by a data source (e.g., from the Twitter API).

**QT-LIB Solution:** We have modified *kafka-perf-tool* to inject records as custom times supplied in the trace file to be replayed. This has been tested against synthetic Twitter data generated by QT-GEN.

- **Baseline limitation 3:** in the QT-LIB library for Storm applications, DICE offers to the end user the possibility to customize all properties of QT-LIB in a programmatic manner, directly from within the Java application code. This was preferred to a solution involving JSON or XML configuration files for each test, which created certain complication in the automated deployment phase of the QT spouts on the application testbed. However, *kafka-perf-tool* follows a JSON-based configuration approach, being primarily designed as a command line tool. Thus it needs to be modified.

**QT-LIB Solution:** To overcome this limitation, we have defined a wrapper API that, similar to what already offered by QT-LIB for Storm, enables the developer to fully control the *kafka-perf-tool* load testing directly from within the Java application code. In this way, a simple unit test for the application can be written to test its performance.

### 2.2.3 Final QT-LIB internal architecture

In light of the above changes, the final internal architecture of QT-LIB is summarized in Figure 2. We assume in this overview that the typical usage consists of replaying a given trace, recorded in a log file. We have the following steps:

- The QT-LIB user should first provide the trace in JSON format to QT-GEN, which will then produce one or more new output traces for subsequent use with QT-LIB. Such output traces provide statistically similar data to the one present in the original log file but typically differ

by rate of arrival of the messages or the total number of messages in each trace. We point to deliverable D5.4 for an introduction to QT-GEN.

- Subsequently, the user writes Java code within her data-intensive application to use QT-LIB. If the purpose of the load injection is to test a Storm topology, the user will need to instantiate a *RateSpout* object, whereas for Kafka, she will need a *RateProducer* object. In both cases, these objects can be instantiated within the DIA code.
- Prior to execution of the test, the output trace generated by QT-GEN to be used in the test needs to be packaged as a resource within the *jar* file of the DIA.
- In the case of Kafka load-testing, it is possible to consume from a Spark or Storm application the Kafka topic that is used by QT-LIB to send messages.

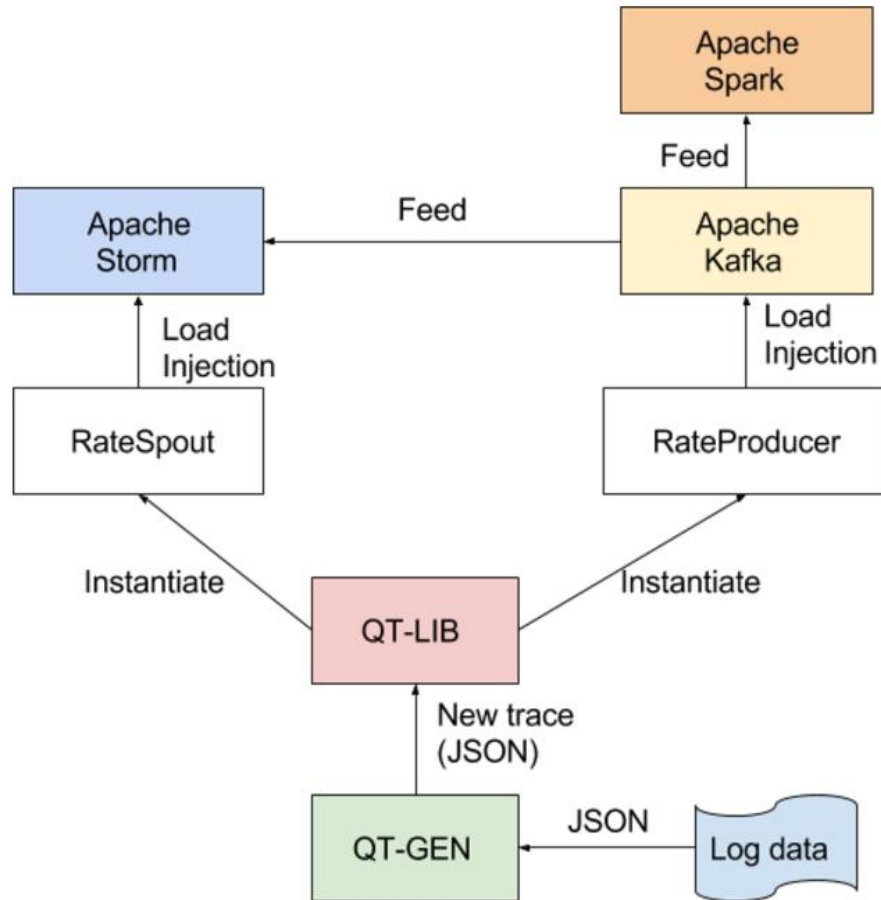


Figure 2 Revised QT-LIB architecture

#### 2.2.4 Using QT-LIB for Kafka

We now illustrate the practical use of QT-LIB with Kafka. Below we present a compact example that is included in the QT-LIB distribution (*KafkaRateProducer.java*). Our goal is to load test a Kafka instance by injecting a set of random messages and JSON messages from a trace file under a specified topic.

Initially, as done also with Storm, we construct a *QTLoadInjector* factory that will assemble the load injector. Moreover, we specify the name of the input trace file, which is assumed to be packaged within the *jar* file of the data-intensive application.

```
public static void main(String[] args) {  
    QTLloadInjector QT = new QTLloadInjector();  
    String input_file = "test.json"; // this is assumed to be a resource of the project jar file
```

---

We are now ready to instantiate a QT load injector for Kafka, which is called a *RateProducer*:

```
RateProducer RP;  
RP=QT.getRateProducer();
```

We assume that our target topic is called *dice3* and it is exposed by a Kafka instance available on the local machine on port 9092. This corresponds to the port of the so-called Kafka bootstrap server, and should not be confused with the port of the Zookeeper instance associated to Kafka. We can use the *run()* command of the *RateProducer* to start immediately the experiment.

```
String topic = "dice3"; // topic get created automatically  
String bootstrap_server = "localhost:9092";  
  
// random message data - default is to send a single message  
RP.run(bootstrap_server, topic);
```

By default, the *RateProducer* will send a single random message of size 1024 bytes to the server. It is possible to increase the number of messages to be sent using the *setMessageCount(int msgCount)* and *setMessageSize(int msgSize)* methods exposed by *RateProducer*.

We may now repeat the same workload generation, but reading the message from the *test.json* file shipped with the application *jar*. To do so, we use the syntax

```
// data from jsonfile  
RP.run(bootstrap_server, topic, input_file);
```

where we now specify the input trace file. In the DICE distribution of QT-LIB, *test.json* contains 100 messages. In order to validate the ability of QT-LIB to cyclically reuse this set of messages we run

```
// data from jsonfile -- this is going to read more data than available in the json file, looping  
RP.setMessageCount(101);  
RP.run(bootstrap_server, topic, input_file);
```

### 2.2.5 Kafka support validation

To validate the extended capabilities of QT-LIB we have performed two major tests. In the first test, we have run QT-LIB against a Kafka installation, producing a sequence of JSON messages from ATC's Social Sensor platform onto a newly created topic, and subsequently using the *kafka-console-consumer* utility, shipped with the default distribution of Kafka, to check that the topic correctly received the messages. This test proved successful and it is included in the *example/* folder of the QT-LIB official distribution on github.

Next, we have considered a more challenging scenario in which we have injected load on a Kafka pipeline, which was later pushed onto a Spark testbed running a wordcount application connected to a MySQL instance. The Spark instance was installed using a simple open source distribution that offers a graphical dashboard to visualize the volume of messages received by wordcount<sup>2</sup>. We modified the test code of QT-LIB to inject into the topic used by this wordcount application by modifying the input data to align with the format used by wordcount.

---

<sup>2</sup> <https://github.com/trK54Ylmz/kafka-spark-streaming-example>



The format of the messages included in the JSON file reproduced by QT-LIB is as follows:

```
{"type":"south_america","value":893.4258}  
{"type":"oceania","value":989.20374}  
{"type":"oceania","value":554.9144}
```

Here *type* and *value* are exemplars of JSON fields, and do not bear a specific meaning for the application's internal logic, which simply counts the number of messages of a given type. Figure 3 below is a screenshot from the dashboard of the wordcount application, showing the rate of message flow in the system for two types of messages sent by QT-LIB ("oceania" and "south\_america"), one injected with constant rate, the other with peak rate in the central part of the experiment.



Figure 3 QT-lib JSON data injection in a Spark testbed for two types of messages. The y-axis represents the number of messages, the x-axis is dynamically updated by the wordcount application every few seconds.

## 2.3 Extended integration with Storm and DMON

### 2.3.1 DMON integration

QT-LIB now offers a new class, called *DMONCapacityMonitor*, which eases the integration of QT with DMON. Let us consider for example the following scenario: a user wishes to increase the load on a Storm testbed until hitting peak capacity at one of the bolts, which will therefore be a bottleneck for the DIA. To avoid making the system unstable, the user wants to *progressively* increase the load until reaching the desired peak capacity, but unfortunately it is not possible to predict beforehand how many messages the system will need to inject before incurring a capacity bottleneck. By using *DMONCapacityMonitor* the DICE user can easily check automatically from DMON if the system has reached the desired utilization.

The usage of *DMONCapacityMonitor* is illustrated in the follow example. The end user first specifies the desired time-window to check in DMON, which in the example corresponds to 30 seconds on 4-March-2017, between 12:19:30 and 12:20:00. As this may still correspond to many records, it is possible to require that at most *maxDMONRecords* are actually retrieved from the monitoring platform, neglecting the others. *DMONCapacityMonitor* exposes a function *getMaxCapacity* that recursively parses the JSON data retrieved from DMON, which is located via the specified URL and port, until determining the maximum capacity utilization across all bolts. The recursion is needed to accommodate for arbitrarily nested JSON files.

```

public static void main(String[] args) {
    try {
        String t0 = "2017-03-04T12:19:30.000Z";
        String timeStampDate = new SimpleDateFormat("yyyy-MM-dd").format(new Date());
        String timeStampTime = new SimpleDateFormat("HH:mm:ss.000").format(new Date());
        String t1 = timeStampDate + "T" + timeStampTime + "Z";
        System.out.println(t1);
        t1 = "2017-03-04T12:20:00.000Z";
        int maxDMONRecords = 100;
        double maxcapacity = getMaxCapacity("http://109.231.122.229:5001", t0, t1, maxDMONRecords);
        System.out.println("Max Bolt Capacity: "+maxcapacity);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

```

The main added value of this class is the recursive method to parse the DMON JSON file. The collection mechanism of metrics such as maximum capacity is rather easy to extend to other metrics.

### 2.3.2 Storm integration

In addition to DMON integration, we have added in QT-LIB native methods to retrieve performance data from the Storm UI interface. Storm UI is a graphical user interface, standard within the Storm release, that allows the end user to check status for a running Storm topology. For example, in the screenshot below one can see that it is possible to check latency and throughput (“Emitted” tuples) of a topology directly from Storm UI.

## Storm UI

### Topology summary

| Name      | Id                     | Status | Uptime | Num workers | Num executors | Num tasks |
|-----------|------------------------|--------|--------|-------------|---------------|-----------|
| WordCount | WordCount-1-1387403806 | ACTIVE | 6m 48s | 3           | 28            | 28        |

### Topology actions

### Topology stats

| Window | Emitted | Transferred | Complete latency (ms) | Acked | Failed |
|--------|---------|-------------|-----------------------|-------|--------|
| 10m 0s | 241680  | 129620      | 0.000                 | 0     | 0      |

The DICE QT class *StormUICapacityMonitor* delivers similar functionalities of *DMONCapacityMonitor* but uses Storm UI as the source of the data. In particular, it also offers a *getMaxCapacity* API to extract the maximum bolt capacity. A usage example is given below.



```
public static void main(String[] args) {
    try {
        String encodedId = getId("http://localhost:8080", "topology-qt1");
        double maxcapacity = getMaxCapacity("http://localhost:8080", encodedId);
        System.out.println("Max Bolt Capacity (StormUI, topology="+encodedId+"): "+maxcapacity);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

As shown in the code snippet, a difference with respect to *DMONCapacityMonitor*, is that *StormUICapacityMonitor* requires to uniquely identify the topology through the `getId()` command, which accepts in input the topology name.

### 3 Fault Injection Tool (FIT)

#### 3.1 Overview

The operation of data intensive applications almost always requires dealing with various failures. Therefore during the development of an application, tests have to be made in order to assess the reliability and resilience of the system. These test the ability of a system to cope with faults and to highlight any vulnerable areas.

The FIT allows users to generate faults on their Virtual Machines, giving them a means to test the resiliency of their installation. Using this approach the designers can use robust testing, highlighting vulnerable areas to inspect before it reaches a commercial environment. Users or application owners can test and understand their application design or deployment in the event of a cloud failure or outage, thus allowing for the mitigation of risk in advance of a cloud based deployment.

#### 3.2 Motivation

Current and projected growth in the big data market provides three distinct targets for the tool. Data Centre owners, cloud service providers and application owners are all potential beneficiaries due to their data intensive requirements. The resilience of the underlying infrastructure is crucial to these areas. Data Centre owners can gauge the stress levels of different parts of their infrastructure and thus offer advice to their customers, address bottlenecks or even adapt the pricing of various levels of assurances.

For developers FIT provides the missing and essential service of evaluating the resiliency and dependability of their applications, which can only be demonstrated in the application's runtime by deliberately introducing faults. By designing the FIT to be a lightweight and versatile tool it is trivial to use it during Continuous Integration or within another tool for running complex failure scenarios. Used in conjunction with other tools not within the scope of this report, FIT could monitor and evaluate the effect of various faults on an application and provide feedback to the developers on application design.

#### 3.3 Design

The FIT can generate VM faults for use by application owners and VM admins. The tool is designed to run independently and externally to any target environment, as indicated in Figure 4.

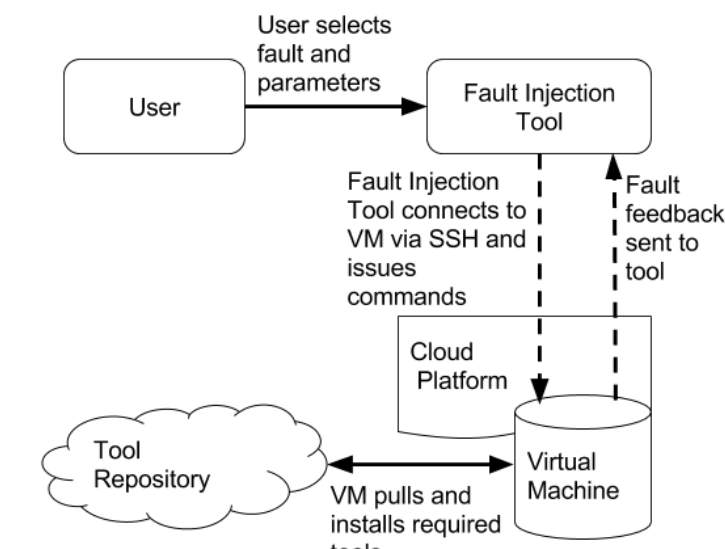


Figure 4 Fault Injection Tool architecture

To access the VM level and issue commands the DICE FIT uses JSCH to SSH to the Virtual Machines. By using JSCH the tool is able to connect to any VM that has SSH enabled and can then issue commands as a pre-defined user. This allows greater flexibility of commands as well as the installation of tools and dependencies. The DICE FIT is released under the permissive APACHE LICENCE 2.0 and supports the OS configurations Ubuntu (tested with versions 14.04 and 15.10), and Centos with set Repo configured and *wget* installed (tested on version 7).

### 3.4 Operation

The FIT is designed to work from the command line or through a Graphical User Interface. The user can invoke actions which connect to the target VM and automatically install any required tools and dependencies or evoke the required APIs. The command line switches and parameters allow users to select a specific fault and the parameters of the fault such as the amount of RAM to user or which services to stop.

An example command line call to connect to a node using SSH and cause memory stress with 2GB is as follows:

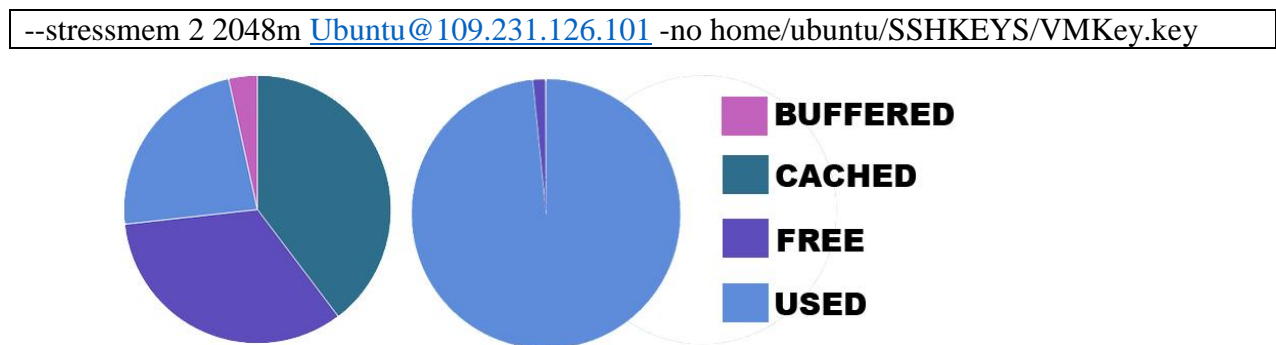


Figure 5 Memory available on the target node before (left) and during the invocation (right)

This call was ran on a real cloud system. The tool connected via SSH and determined the OS version by checking the `/etc/*-release`, Ubuntu in this case. It then gathered the memory stress tool suitable for Ubuntu, which is Memtester in this case. Finally the FIT called Memtester to saturate memory on the target node. Figure 5 shows the results as detecting by a monitoring tool, where it can be seen that nearly all 2GB of available RAM had been saturated.

### 3.5 Graphical User Interface

The GUI provides the same functionality as the command line version of the tool. The GUI provides users with a visual way of interacting with the tool which can make the tool more accessible for a range of users. The user can select from the available actions from a home screen, as seen in Figure 6. Each button leads to a page where a user can enter the relevant inputs and then the fault can be executed. These inputs are the equivalent of the command line parameters.


|  |  |
|--|--|
| <b>Virtual Machine Failures</b>  |  |
| <b>Deployment Faults</b>   | Create faults within a deployment                  |
| <b>Block a VM</b>  | Block external access to a virtual machine         |
| <b>Stop Random VM</b>  | Stop a random VM from a cloud in FCO               |
| <b>Stop a Service</b>  | Stop a service running on a virtual machine        |
| <b>Virtual Machine Resource Overload</b>   |  |
| <b>CPU Overload</b>  | Overload the CPU, RAM or Disk on a virtual machine |
| <b>RAM Overload</b>  |  |
| <b>Disk Overload</b>   |  |

Figure 6 The FIT GUI home screen

In Figure 7 we can see the CPU overload page where the user enters the details of the VM and the amount of time to run the overload for. In place of the password the user can also upload an SSH key from a file. Any feedback and output from running the fault is shown at the bottom of the page.


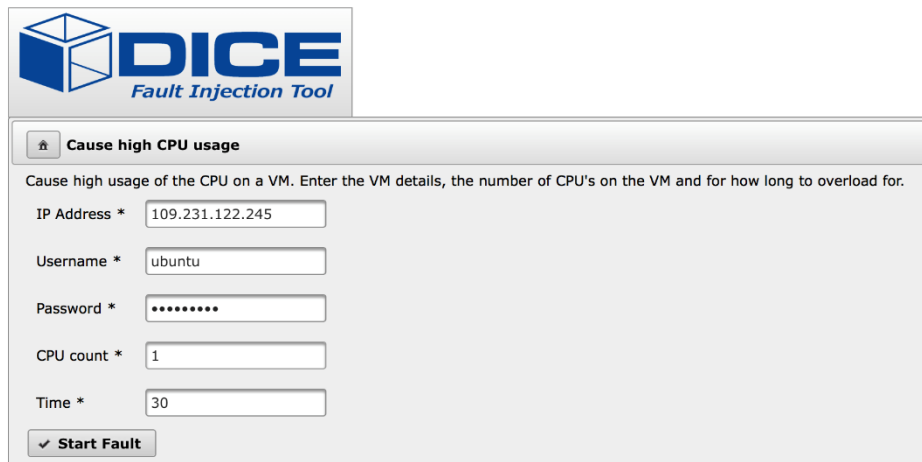
|   |   |
|--|---|
| <b>CPU Overload</b>  |   |
| Cause high usage of the CPU on a VM. Enter the VM details, the number of CPU's on the VM and for how long to overload for. |   |
| IP Address *   | <input type="text"/>                    |
| Username *   | <input type="text"/>                    |
| Password *   | <input type="password"/>                |
| CPU count *  | <input type="text"/>                    |
| Time *   | <input type="text"/>                    |
| <input checked="" type="checkbox"/> <b>Start Fault</b>   |   |
| Upload an SSH key in place of the password   |   |
| <input type="button" value="+ Choose"/>  | <input type="button" value="✓ Submit"/> |
| <b>Output</b>  |   |
| Output: Globalfirst  |   |

Figure 7 CPU Overload page

Using the GUI is a straightforward process of selecting the desired fault and providing the VM details. In the example below a high CPU usage fault is chosen, and the address, username and password of the VM is entered. The number of CPUs on the machine is entered and then the amount of time to overload the CPU for, in this case 30 seconds, as shown in Figure 8.



The screenshot shows the DICE Fault Injection Tool (FIT) GUI. At the top is the DICE logo with the text 'Fault Injection Tool'. Below the logo is a tab labeled 'Cause high CPU usage'. Under the tab, there is a descriptive text: 'Cause high usage of the CPU on a VM. Enter the VM details, the number of CPU's on the VM and for how long to overload for.' Below this text are several input fields: 'IP Address \*' with the value '109.231.122.245', 'Username \*' with the value 'ubuntu', 'Password \*' with masked characters '\*\*\*\*\*', 'CPU count \*' with the value '1', and 'Time \*' with the value '30'. At the bottom of the form is a button labeled 'Start Fault' with a checkmark icon.

Figure 8 High CPU usage GUI input

### 3.1 Integration

A major step forward in the development of the Fault Injection Tool is the integration with other DICE tools. This work incorporates the FIT deeper within the DICE toolset, and provides further useful functionality for users of the FIT.

One tool in which the FIT has been fully integrated with is the DICE Deployment Service, developed by XLAB, which is shown in Figure 9. This new feature allows faults to be caused on all of the VMs which make up a deployment into a DICE virtual deployment container.

The method in which this integration works is through the GUI version of the FIT. First, the user must acquire a token from the deployment service, in order to be able to authenticate with the API. From there, an option is given on the GUI to list all containers running on the DICE deployment service. The user can then choose the container they wish to cause faults on. A JSON file can also be uploaded in order to further customise the type of faults to be caused on certain VMs within the deployment. This is accomplished by matching a fault with the name of the component type that is associated (e.g., hosted on) with the VM in the application's deployment blueprint. After these attributes are provided, the desired faults are automatically caused on all of the selected VMs inside the container, to simulate the faults occurring at an application level. This enables that the user needs to fill in the form only once for a virtual deployment container, then use the same information for all the subsequent (re)deployments in the container.

Additionally, we show in the next chapter the integration of FIT with the new *dmon-gen* utility to automate the generation of anomalies.

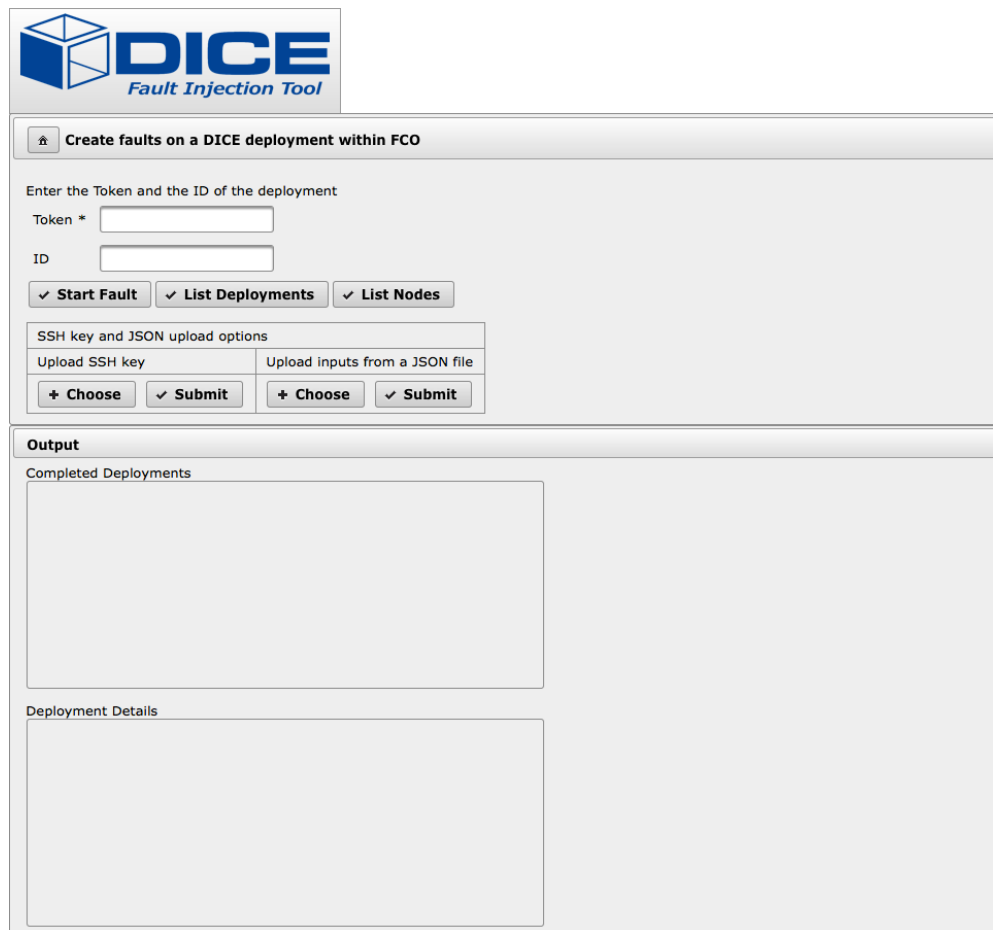


Figure 9 Fault Injection tool Dice Deployment Service

### 3.2 Validation

Using the Linux ‘top’ command on the target VM the current state of its resources can be seen. Before running the CPU overload fault, the %Cpu usage is at 0.7% as seen in Figure 10. While running the CPU overload the %Cpu quickly rises to 100%. In the processes below we can see that the stress command is using 99.2% of the CPU as shown in Figure 11.

```
top - 08:33:21 up 22:27, 2 users, load average: 0.07, 0.04, 0.05
Tasks: 67 total, 1 running, 66 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.7 us, 1.0 sy, 0.0 ni, 98.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 501772 total, 347048 used, 154724 free, 51092 buffers
KiB Swap: 0 total, 0 used, 0 free. 231424 cached Mem
```

| PID | USER | PR | NI | VIRT  | RES  | SHR  | S | %CPU | %MEM | TIME+   | COMMAND     |
|-----|------|----|----|-------|------|------|---|------|------|---------|-------------|
| 1   | root | 20 | 0  | 33640 | 2896 | 1444 | S | 0.3  | 0.6  | 0:10.01 | init        |
| 69  | root | 20 | 0  | 0     | 0    | 0    | S | 0.3  | 0.0  | 0:43.09 | kworker/0:2 |

Figure 10 Before running CPU overload

```
top - 08:36:28 up 22:30, 2 users, load average: 0.22, 0.07, 0.06
Tasks: 73 total, 3 running, 70 sleeping, 0 stopped, 0 zombie
%Cpu(s):100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 501772 total, 350672 used, 151100 free, 51120 buffers
KiB Swap: 0 total, 0 used, 0 free. 231592 cached Mem
```

| PID   | USER   | PR | NI | VIRT  | RES  | SHR  | S | %CPU | %MEM | TIME+   | COMMAND |
|-------|--------|----|----|-------|------|------|---|------|------|---------|---------|
| 14533 | ubuntu | 20 | 0  | 7304  | 96   | 0    | R | 99.2 | 0.0  | 0:12.57 | stress  |
| 14417 | ubuntu | 20 | 0  | 23536 | 1500 | 1092 | R | 0.7  | 0.3  | 0:00.56 | top     |

Figure 11 While running CPU overload

The GUI successfully complements the objectives achieved by the command line tool. It makes the fault injection tool highly accessible, allowing anyone to find vulnerabilities and test the resiliency of their systems.

### **3.3 Installation**

The source code can be found in the DICE GitHub repository<sup>3</sup>. The repository contains the source code and a WAR file so it can be deployed on a server, such as Apache Tomcat. Once the image is deployed on the server it will be immediately available and ready to use.

---

<sup>3</sup> <https://github.com/dice-project/DICE-Fault-Injection-GUI>

## 4 DMON-Gen Utility

### 4.1 General Overview

The proliferation of Big Data technologies and of the DIAs based on these have resulted in a shortage of software developers and architects with specialized knowledge. In particular, the identification of anomalous behavior of these applications differs from one version to another. In DICE we have developed a specialized tool that handles the detection of both point and contextual anomalies. In this deliverable we present a module of the quality testing tool that is suitable to assess anomaly detection performance. In addition to this *DMON-gen* can be used in combination with FIT to run different testing scenarios geared towards ensuring DIA is working as intended.

The Anomaly detection tool (ADT) is detailed in D4.4 and D4.5 respectively. For the sake of completeness we will give a short description in the following paragraphs. One of the most important facts to consider is that the ADT has a closer integration with DMON than any other tools from the DICE solution. That is mainly due to two facts. Firstly, ADT needs data on which to run anomaly detection methods. Thus it is extremely important to have data available in a format which is usable. Second, ADT together with the monitoring forms a lambda architecture. Each instance of ADT can have the role of batch or speed layer while DMON has the role of a serving layer.

As mentioned before the detected anomalies will be sent and indexed into DMON. All DICE actors and tools will be able to query this special index to see/listen for detected anomalies. In this way it is possible to create specialized ADT instances for each anomaly detection method in part. The result will be reflected in the same index from DMON. This architecture also allows us to serve the results of both the monitoring and anomaly detection on the same endpoint (DMON).

As mentioned in section some anomaly detection methods, more precisely the ones using supervised learning techniques, need labelled data in order to function properly. The use of supervised learning for this task is a fairly complicated thing to accomplish. One solution is to label all normal data instances and all unlabelled instances are considered anomalies. As observed in most systems the normal data instances outnumber by far the anomalous ones, so labelling them manually is extremely impractical.

It is easy to see that we require a way in which we can create semi-automatically labelled training data that can be used by ADT in order to create predictive and cluster models. The resulting tool, called *DMON-gen*, is able to execute several jobs on Big Data platforms such as Yarn or Spark based on user defined parameters.

### 4.2 Architecture

End users are able to define experiments which in themselves are made up of a series of jobs. A job contains both runtime parameters for the DIAs as well as platform specific parameters. *DMON-gen* has to be located on one of the hosts which comprise a DIA deployment. It is not necessary for it to be on a particular host as long as it is able to execute jobs from it.

This allows users not only to specify different jobs that need executing but to create a particular



usage pattern or load when it comes to Yarn and Spark based systems.

### 4.3 Monitoring data generation

In order to use *DMON-gen* the user is required to create the experiment description as mentioned in the previous sections. This descriptor is in JSON format. An example can be seen in Listing 1.

```
{
  "exp1":
  [
    {
      "yarn":["pi","10","100"],
      "cardinality": 1,
      "conf":{"hdfs":{"DATANODE": {"dfs_datanode_du_reserved": "8455053312"}},
        "yarn":{"NODEMANAGER": {"mapreduce_am_max-attempts": "2"}}}
    }
  ]
}
```

**Listing 1 Experiment Descriptor for DMON-gen**

We see that the descriptor defines one YARN experiment called *pi* which has two parameters (10 signifies the number of maps while 100 is the sample size). The *cardinality* setting is used to define how often the experiment is to be run. Some experiments might require the execution of the same DIA numerous times without changing any settings. In this case *pi* is the DIA which runs on Yarn. Because *DMON-gen* has no preconceptions about the DIA it has to run (it is application agnostic) we only need to specify any command line parameter that might be required. Of course these command line parameters could also point to a configuration file. It is up to the end user and developer to decide which ones are the required parameters. If a DIA does not require any parameters, only the application type (in his case “yarn”) and DIA name (in the example it is “pi”) has to be defined. Lastly, we have the *conf* settings which denote the settings based on roles for each big data service. It is that we use Cloudera CDH 5.7.x for *DMON-gen* so the naming conventions are the same for our tool as with the current version of CDH<sup>8</sup>.

Listing 1 shows that we wish to change the configuration of the HDFS service data-node roles reserved space for non DFS and for Yarn services the node manager roles maximum number of jobs attempts value. The complete list of available parameters can be found in Annex of this document.

Inspiration and starting point for the creation of this tool was the generation of semi-labelled training data for the anomaly detection platform. It is easy to see that by using *DMON-gen* we are able to induce some types of anomalies in an automatic manner and are then be able to correlate these with the metrics collected during a specified time-frame. In essence labelling the data based on the settings from both the DIA parameters and platform specific parameters. Once a particular experiment defined in *DMON-gen* has run its course we can use the output to label metrics data from the monitoring platform. For example we have set some parameters related to Yarn mappers so we can see from the *DMON-gen* output at what time each map task has run. We use this information to add a label in the monitoring data.

During platform specific parameter changes the entire infrastructure might need to be restarted. In these situations *DMON-gen* will not only enact the changes but also enforce them by checking the correct application of the new parameters. Once a restart is needed it will wait for the platform to come back online and then start the execution of the DIA. Invalid parameters are not caught before execution but rather at execution start. If this happens *DMON-gen* will just skip the offending job and start executing the next set of jobs.

ADT<sup>4</sup> as well as the *DMON-gen* tool can be found at the official DICE Github Repository. These repositories also contain the up to date documentation for each of the tools.

#### 4.4 Use-cases and configuration

All experiments for DMON load testing and ADT functionality testing was done using *DMON-gen*. This setup allowed us to define a set of long running experiments that took days to weeks to run. As soon as they finished to just collect the data from DMON. The example given below illustrates the combined use of *DMON-gen* with the Fault Injection tool (FIT).

```
{
  "exp1":
  [
    {
      "yarn":["pi","10000","10000000"],
      "cardinality": 1,
      "conf":{"hdfs":{"DATANODE":{"dfs_datanode_du_reserved": "8455053312"}}},
      "yarn":{"NODEMANAGER":{"mapreduce_am_max-attempts": "2"}}},
      "fit":["cpu", "mem"]
    },
    {
      "yarn":["pi", "1000", "10000"],
      "cardinality": 1,
      "conf":{"hdfs":{"DATANODE":{"dfs_datanode_du_reserved": "8455053"}}},
      "yarn":{"NODEMANAGER":{"mapreduce_am_max-attempts": "5"}}}
    },
    {
      "yarn":["pi", "10000", "1000"],
      "cardinality": 1,
      "conf":{"hdfs":{"DATANODE":{"dfs_datanode_du_reserved": "845"}}},
      "yarn":{"NODEMANAGER":{"mapreduce_am_max-attempts": "5"}}}
    }
  ],
  "exp2":
  [
    {
      "spark":["pi", "100000000"],
      "cardinality": 100
    },
  ],
}
```

<sup>4</sup> <https://github.com/dice-project/DICE-Anomaly-Detection-Tool>

```
{  
  "spark":["pi", "10000"],  
  "cardinality": 100  
}  
  
]
```

#### Listing 2 Experiment Descriptor example

Listing 2 details one of the experiment batches. We can see that it defines 2 experiments. In the first experiment the “*pi*” DIA is run three times. Each job changes data node and node manager related parameters. The second experiment contains 2 spark jobs which run 100 times each.

We can also specify Fault Injection Tool (FIT) specific parameters for each experiment. In Listing 2 we can see that “*fit*” is set to stress the CPU and memory of all of the hosts from the platform. If not otherwise specified FIT is set to stress CPU and memory to 50% of the available resources.

Platform specific parameters are not always required to be explicitly set. Most of the time during development DIA parameters are much more important. Because of this platform specific parameters are not mandatory to be defined in *DMON-gen*, it may be sufficient to specify the yarn and cardinality configuration to be able to run experiments. The tool enables developers to define different experimental scenarios and see how individual changes are reflected (and detected in the case of ADT) by the available metrics.

## 4.5 Validation

The above sections detail how to setup the job descriptor, in this section we will show what the output is and how we use it to create a labeled dataset. The *DMON-get* tool has to be located on one of the processing nodes that houses the DIA. The user then decides what job he or she wants to run and what parameters to use.

The output of *DMON-gen* can be split up into 3 distinct parts. The first part (see Listing 3) shows the main output containing high level data of the job that is executed. It gives us the name of the experiments as well as the beginning and end timestamp of each run together with the settings used.

The second part of the output has information on each of the jobs from the first output. It is in fact a modified version of the debug output from big data platform services (i.e. Spark, Yarn, HDFS etc). These contain fine grained information of the currently running jobs. The most useful data is that pertaining to the internal state and metrics associated with the currently running job. We can see exactly when a map or reduce phase was executed, how long it took, the number of bytes written/read etc. You can see an example output in Listing 4.

The last part of the output is the actual monitoring data resulting from all of the jobs defined in *DMON-gen*. The quantity of information contained depends on how long the jobs took to execute and on the polling period set during DMON setup.

```

Loaded experimental descriptor from pieexp.json
Started jobs at 2017-03-16 07:07:49.882789
Started experiment pierunexperiment1 at 2017-03-16 07:07:49.882947
Started iteration 0 for job {u'cardinality': 5, u'yarn': [u'pi', u'10', u'100']} from experiment pierunexperiment1 at 2017-03-16
07:07:49.883308
Yarn job selected with arguments [u'pi', u'10', u'100']
Started job exp-pierunexperiment1-0-1
Finished iteration 0 for job {u'cardinality': 5, u'yarn': [u'pi', u'10', u'100']} from experiment pierunexperiment1 at 2017-03-16
07:08:22.729229
Started iteration 1 for job {u'cardinality': 5, u'yarn': [u'pi', u'10', u'100']} from experiment pierunexperiment1 at 2017-03-16
07:08:22.729531
Yarn job selected with arguments [u'pi', u'10', u'100']
Started job exp-pierunexperiment1-1-1
Finished iteration 1 for job {u'cardinality': 5, u'yarn': [u'pi', u'10', u'100']} from experiment pierunexperiment1 at 2017-03-16
07:08:53.649757
Started iteration 2 for job {u'cardinality': 5, u'yarn': [u'pi', u'10', u'100']} from experiment pierunexperiment1 at 2017-03-16
07:08:53.650065
Yarn job selected with arguments [u'pi', u'10', u'100']
Started job exp-pierunexperiment1-2-1
Finished iteration 2 for job {u'cardinality': 5, u'yarn': [u'pi', u'10', u'100']} from experiment pierunexperiment1 at 2017-03-16
07:09:24.198293
Started iteration 3 for job {u'cardinality': 5, u'yarn': [u'pi', u'10', u'100']} from experiment pierunexperiment1 at 2017-03-16
07:09:24.198600
Yarn job selected with arguments [u'pi', u'10', u'100']
...

```

**Listing 3 Example of DMON-gen first level output**

```

17/03/16 07:07:55 INFO mapreduce.Job: Running job: job_1489577180719_0001
17/03/16 07:08:04 INFO mapreduce.Job: Job job_1489577180719_0001 running in uber mode : false
17/03/16 07:08:04 INFO mapreduce.Job: map 0% reduce 0%
17/03/16 07:08:10 INFO mapreduce.Job: map 10% reduce 0%
17/03/16 07:08:12 INFO mapreduce.Job: map 30% reduce 0%
17/03/16 07:08:14 INFO mapreduce.Job: map 70% reduce 0%
17/03/16 07:08:15 INFO mapreduce.Job: map 80% reduce 0%
17/03/16 07:08:18 INFO mapreduce.Job: map 100% reduce 0%
17/03/16 07:08:22 INFO mapreduce.Job: map 100% reduce 100%
17/03/16 07:08:22 INFO mapreduce.Job: Job job_1489577180719_0001 completed successfully
17/03/16 07:08:22 INFO mapreduce.Job: Counters: 49
    File System Counters
        FILE: Number of bytes read=94
        FILE: Number of bytes written=1293891
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=2700
        HDFS: Number of bytes written=215
        HDFS: Number of read operations=43
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=3
    Job Counters
        Launched map tasks=10
        Launched reduce tasks=1
        Data-local map tasks=10
        Total time spent by all maps in occupied slots (ms)=53924
        Total time spent by all reduces in occupied slots (ms)=4058
        Total time spent by all map tasks (ms)=53924
        Total time spent by all reduce tasks (ms)=4058
        Total vcore-seconds taken by all map tasks=53924
        Total vcore-seconds taken by all reduce tasks=4058
        Total megabyte-seconds taken by all map tasks=55218176
        Total megabyte-seconds taken by all reduce tasks=4155392
    Map-Reduce Framework
        Map input records=10
        Map output records=20
        Map output bytes=180
        Map output materialized bytes=340

```

```
Input split bytes=1520
Combine input records=0
Combine output records=0
Reduce input groups=2
Reduce shuffle bytes=340
Reduce input records=20
Reduce output records=0
Spilled Records=40
Shuffled Maps =10
Failed Shuffles=0
Merged Map outputs=10
GC time elapsed (ms)=1132
CPU time spent (ms)=9080
Physical memory (bytes) snapshot=4698320896
Virtual memory (bytes) snapshot=28443156480
Total committed heap usage (bytes)=4822401024

Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0

File Input Format Counters
  Bytes Read=1180
File Output Format Counters
  Bytes Written=97
```

**Listing 4 Example of DMON-gen second level output**

These two files contain enough information to label the monitoring data. We can label based on job or even at discrete task level (i.e. metrics of a particular map task). This runtime information together with the parameters defined in the description file make it easy to link each parameter changes to the resulting concrete metrics.

## 5 Conclusion

### 5.1 Summary of achievements

In this deliverable we have presented the final version of the quality testing tools (QT) and the fault injection tool (FIT). The main advancements in year 3 are novel support for Kafka and Spark load testing, a utility for injecting anomalies in a data-intensive application (*DMON-gen*), and a GUI for the fault injection tool. In Section 2 we provided a summary of the requirements. Table 1 below indicates the level that DICE QT comply in their final release. The *Level of fulfilment* column has the following values:

- X - not supported in the initial version yet
- ✓ - initial support
- ✓✓ - medium level support
- ✓✓✓ - fully supported

**Table 1:** Level of compliance of the initial version of the DICE delivery tools with the initial set of requirements.

| Requirement    | Title   | Priority | Level of fulfilment |
|----------------|---|----------|---------------------|
| <b>R5.6</b>    | Test workload generation  | MUST     | ✓✓✓                 |
| <b>R5.8.2</b>  | Starting the quality testing  | MUST     | ✓✓✓                 |
| <b>R5.8.3</b>  | Test run independence   | MUST     | ✓✓✓                 |
| <b>R5.8.5</b>  | Test outcome  | MUST     | ✓✓✓                 |
| <b>R5.13</b>   | Test the application for efficiency   | MUST     | ✓✓✓                 |
| <b>R5.14.1</b> | Test the behavior when resources become exhausted   | MUST     | ✓✓✓                 |
| <b>R5.17</b>   | Quick testing vs comprehensive testing  | MUST     | ✓✓✓                 |
| <b>R5.7</b>    | Data loading support  | SHOULD   | ✓✓                  |
| <b>R5.7.2</b>  | Data feed actuator  | SHOULD   | ✓✓✓                 |
| <b>R5.14.2</b> | Trigger deliberate outages and problems to assess the application's behavior under faults | SHOULD   | ✓✓✓                 |

As shown in the table, in year 3 we have fulfilled all the main requirements for the quality testing and fault injection. The main advancements compared to year 2 of the project are:

- The greater integration with the rest of DICE (R5.8.2, R5.8.5), which allows to start/stop and visualize the outcome of the experiments using the DICE Delivery Tools. These advances of QT are reported primarily in deliverables D5.3 and D6.3.
- The ability to control the experiment duration and quickly or comprehensively test Storm, Spark, and Kafka based DIAs (R5.17), loading data from external sources such as MongoDB or JSON files (R5.7), producing test workloads with QT-GEN (R5.6), and feed the loaded data into the system using QT-LIB (R5.7.2).
- The FIT and *DMON-gen* tools fulfills the requirements of R5.14.2 and R5.17, by providing to the user the ability to generate faults in their data-intensive applications.

## References

- [1] Deliverable D1.2 - Requirement specification. Available from: <https://www.dice-h2020.eu/deliverables/>

## 6 Annex

### 6.1 Parameter settings for DMON-gen

#### 6.1.1 Parameters for HDFS

Configuration parameters for HDFS service type. It is important to note that name node and secondary name node role parameters have the same name. During normal performance evaluation and testing secondary name node parameter setting are irrelevant.

| Role      | Display Name   | API Name   |
|-----------|--|--|
| Data Node | DataNode Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml | datanode_config_safety_valve                     |
|           | Java Configuration Options for DataNode                                  | datanode_java_opts                               |
|           | Available Space Policy Balanced Preference                               | dfs_datanode_available_space_balanced_preference |
|           | Available Space Policy Balanced Threshold                                | dfs_datanode_available_space_balanced_threshold  |
|           | DataNode Volume Choosing Policy  | dfs_datanode_volume_choosing_policy              |
|           | Hadoop Metrics2 Advanced Configuration Snippet (Safety Valve)            | hadoop_metrics2_safety_valve                     |
|           | DataNode Logging Advanced Configuration Snippet (Safety Valve)           | log4j_safety_valve                               |
|           | Heap Dump Directory  | oom_heap_dump_dir                                |
|           | Dump Heap When Out of Memory   | oom_heap_dump_enabled                            |
|           | Kill When Out of Memory  | oom_sigkill_enabled                              |
|           | Automatically Restart Process  | process_auto_restart                             |
|           | DataNode Data Directory  | dfs_data_dir_list                                |
|           | Reserved Space for Non DFS Use   | dfs_datanode_du_reserved                         |
|           | DataNode Failed Volumes Tolerated  | dfs_datanode_failed_volumes_tolerated            |
|           | DataNode Balancing Bandwidth   | dfs_balance_bandwidthPerSec                      |
|           | Enable purging cache after reads   | dfs_datanode_drop_cache_behind_reads             |
|           | Enable purging cache after writes  | dfs_datanode_drop_cache_behind_writes            |
|           | Handler Count  | dfs_datanode_handler_count                       |



|   |                                    |
|---|------------------------------------|
| Maximum Number of Transfer Threads  | dfs_datanode_max_xcievers          |
| Number of read ahead bytes  | dfs_datanode_readahead_bytes       |
| Enable immediate enqueueing of data to disk after writes                            | dfs_datanode_sync_behind_writes    |
| Hue Thrift Server Max Threadcount   | dfs_thrift_threads_max             |
| Hue Thrift Server Min Threadcount   | dfs_thrift_threads_min             |
| Hue Thrift Server Timeout   | dfs_thrift_timeout                 |
| Maximum Process File Descriptors  | rlimit_fds                         |
| Bind DataNode to Wildcard Address   | dfs_datanode_bind_wildcard         |
| DataNode HTTP Web UI Port   | dfs_datanode_http_port             |
| Secure DataNode Web UI Port (SSL)   | dfs_datanode_https_port            |
| DataNode Protocol Port  | dfs_datanode_ipc_port              |
| DataNode Transceiver Port   | dfs_datanode_port                  |
| Use DataNode Hostname   | dfs_datanode_use_datanode_hostname |
| Java Heap Size of DataNode in Bytes   | datanode_java_heapsize             |
| Maximum Memory Used for Caching   | dfs_datanode_max_locked_memory     |
| Cgroup CPU Shares   | rm_cpu_shares                      |
| Cgroup I/O Weight   | rm_io_weight                       |
| Cgroup Memory Hard Limit  | rm_memory_hard_limit               |
| Cgroup Memory Soft Limit  | rm_memory_soft_limit               |
| Java Configuration Options for Failover Controller                                  | failover_controller_java_opts      |
| Failover Controller Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml | fc_config_safety_valve             |
| Failover Controller Logging Advanced Configuration Snippet (Safety Valve)           | log4j_safety_valve                 |
| Heap Dump Directory   | oom_heap_dump_dir                  |
| Dump Heap When Out of Memory  | oom_heap_dump_enabled              |
| Kill When Out of Memory   | oom_sigkill_enabled                |
| Automatically Restart Process   | process_auto_restart               |
| Java Heap Size of JournalNode in Bytes  | journalNode_java_heapsize          |

|           |   |  |
|-----------|---|--|
|           | Cgroup CPU Shares   | rm_cpu_shares  |
|           | Cgroup I/O Weight   | rm_io_weight   |
|           | Cgroup Memory Hard Limit  | rm_memory_hard_limit                                   |
|           | Cgroup Memory Soft Limit  | rm_memory_soft_limit                                   |
| Name Node | NFS Gateway Logging Advanced Configuration Snippet (Safety Valve)           | log4j_safety_valve                                     |
|           | NFS Gateway Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml | nfsgateway_config_safety_valve                         |
|           | Java Configuration Options for NFS Gateway                                  | nfsgateway_java_opts                                   |
|           | Heap Dump Directory   | oom_heap_dump_dir                                      |
|           | Dump Heap When Out of Memory  | oom_heap_dump_enabled                                  |
|           | Kill When Out of Memory   | oom_sigkill_enabled                                    |
|           | Automatically Restart Process   | process_auto_restart                                   |
|           | Maximum Process File Descriptors  | rlimit_fds   |
|           | Java Heap Size of NFS Gateway in Bytes                                      | nfsgateway_java_heapsize                               |
|           | Cgroup CPU Shares   | rm_cpu_shares  |
|           | Cgroup I/O Weight   | rm_io_weight   |
|           | Cgroup Memory Hard Limit  | rm_memory_hard_limit                                   |
|           | Cgroup Memory Soft Limit  | rm_memory_soft_limit                                   |
|           | Enable Automatic Failover   | autofailover_enabled                                   |
|           | NameNode Nameservice  | dfs_federation_namenode_nameservice                    |
|           | Invalidate Work Percentage Per Iteration                                    | dfs_namenode_invalidate_work_pct_per_iteration         |
|           | Quorum-based Storage Journal name   | dfs_namenode_quorum_journal_name                       |
|           | Replication Work Multiplier Per Iteration                                   | dfs_namenode_replication_work_multiplier_per_iteration |
|           | Hadoop Metrics2 Advanced Configuration Snippet (Safety Valve)               | hadoop_metrics2_safety_valve                           |
|           | NameNode Logging Advanced Configuration Snippet (Safety Valve)              | log4j_safety_valve                                     |
|           | NameNode Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml    | namenode_config_safety_valve                           |

|  |                                     |
|--|-------------------------------------|
| NameNode Advanced Configuration Snippet (Safety Valve) for dfs_hosts_allow.txt   | namenode_hosts_allow_safety_valve   |
| NameNode Advanced Configuration Snippet (Safety Valve) for dfs_hosts_exclude.txt | namenode_hosts_exclude_safety_valve |
| Java Configuration Options for NameNode  | namenode_java_opts                  |
| Mountpoints  | nameservice_mountpoints             |
| Heap Dump Directory  | oom_heap_dump_dir                   |
| Dump Heap When Out of Memory   | oom_heap_dump_enabled               |
| Kill When Out of Memory  | oom_sigkill_enabled                 |
| Automatically Restart Process  | process_auto_restart                |
| NameNode Handler Count   | dfs_namenode_handler_count          |
| NameNode Service Handler Count   | dfs_namenode_service_handler_count  |
| Hue Thrift Server Max Threadcount  | dfs_thrift_threads_max              |
| Hue Thrift Server Min Threadcount  | dfs_thrift_threads_min              |
| Hue Thrift Server Timeout  | dfs_thrift_timeout                  |
| Maximum Process File Descriptors   | rlimit_fds                          |
| Java Heap Size of Namenode in Bytes  | namenode_java_heapsize              |
| Cgroup CPU Shares  | rm_cpu_shares                       |
| Cgroup I/O Weight  | rm_io_weight                        |
| Cgroup Memory Hard Limit   | rm_memory_hard_limit                |
| Cgroup Memory Soft Limit   | rm_memory_soft_limit                |
| SecondaryNameNode Nameservice  | dfs_secondarynamenode_nameservice   |
| Hadoop Metrics2 Advanced Configuration Snippet (Safety Valve)                    | hadoop_metrics2_safety_valve        |
| SecondaryNameNode Logging Advanced Configuration Snippet (Safety Valve)          | log4j_safety_valve                  |
| Heap Dump Directory  | oom_heap_dump_dir                   |
| Dump Heap When Out of Memory   | oom_heap_dump_enabled               |
| Kill When Out of Memory  | oom_sigkill_enabled                 |
| Automatically Restart Process  | process_auto_restart                |

|  |   |                                       |
|--|---|---------------------------------------|
|  | SecondaryNameNode Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml | secondarynamenode_config_safety_valve |
|  | Java Configuration Options for Secondary NameNode                                 | secondarynamenode_java_opts           |

## 6.2 Parameters for Yarn service

| Roles   | Display Name   | API Name                               |
|---------|--|--|
| Gateway | MapReduce Client Advanced Configuration Snippet (Safety Valve) for mapred-site.xml         | mapreduce_client_config_safety_valve   |
|         | Gateway Client Environment Advanced Configuration Snippet for hadoop-env.sh (Safety Valve) | mapreduce_client_env_safety_valve      |
|         | Client Java Configuration Options  | mapreduce_client_java_opts             |
|         | YARN Client Advanced Configuration Snippet (Safety Valve) for yarn-site.xml                | yarn_client_config_safety_valve        |
|         | Compression Level of Codecs  | zlib_compress_level                    |
|         | Alternatives Priority  | client_config_priority                 |
|         | Client Failover Sleep Base Time  | client_failover_sleep_base             |
|         | Client Failover Sleep Max Time   | client_failover_sleep_max              |
|         | Running Job History Location   | hadoop_job_history_dir                 |
|         | SequenceFile I/O Buffer Size   | io_file_buffer_size                    |
|         | I/O Sort Factor  | io_sort_factor                         |
|         | I/O Sort Memory Buffer (MiB)   | io_sort_mb                             |
|         | I/O Sort Spill Percent   | io_sort_spill_percent                  |
|         | Use Compression on Map Outputs   | mapred_compress_map_output             |
|         | Compression Codec of MapReduce Map Output  | mapred_map_output_compression_codec    |
|         | Map Tasks Speculative Execution  | mapred_map_tasks_speculative_execution |

|  |   |   |
|--|---|---|
|  | Compress MapReduce Job Output                       | mapred_output_compress                      |
|  | Compression Codec of MapReduce Job Output           | mapred_output_compression_codec             |
|  | Compression Type of MapReduce Job Output            | mapred_output_compression_type              |
|  | Default Number of Parallel Transfers During Shuffle | mapred_reduce_parallel_copies               |
|  | Number of Map Tasks to Complete Before Reduce Tasks | mapred_reduce_slowstart_completed_maps      |
|  | Default Number of Reduce Tasks per Job              | mapred_reduce_tasks                         |
|  | Reduce Tasks Speculative Execution                  | mapred_reduce_tasks_speculative_execution   |
|  | Mapreduce Submit Replication                        | mapred_submit_replication                   |
|  | Mapreduce Task Timeout                              | mapred_task_timeout                         |
|  | MR Application Environment                          | mapreduce_admin_user_env                    |
|  | MR Application Classpath                            | mapreduce_application_classpath             |
|  | Shared Temp Directories                             | mapreduce_cluster_temp_dir                  |
|  | Application Framework                               | mapreduce_framework_name                    |
|  | JobTracker MetaInfo Maxsize                         | mapreduce_jobtracker_split_metainfo_maxsize |
|  | Map Task Java Opts Base                             | mapreduce_map_java_opts                     |
|  | Reduce Task Java Opts Base                          | mapreduce_reduce_java_opts                  |
|  | Max Shuffle Connections                             | mapreduce_shuffle_max_connections           |
|  | ApplicationMaster Java Opts Base                    | yarn_app_mapreduce_am_command_opts          |
|  | Job Counters Limit                                  | mapreduce_job_counters_limit                |
|  | Enable Ubertask Optimization                        | mapreduce_job_ubertask_enabled              |
|  | Ubertask Maximum Job Size                           | mapreduce_job_ubertask_maxbytes             |
|  | Ubertask Maximum Maps                               | mapreduce_job_ubertask_maxmaps              |
|  | Ubertask Maximum Reduces                            | mapreduce_job_ubertask_maxreduces           |
|  | Client Java Heap Size in Bytes                      | mapreduce_client_java_heapsize              |
|  | Map Task CPU Virtual Cores                          | mapreduce_map_cpu_vcores                    |
|  | Map Task Maximum Heap Size                          | mapreduce_map_java_opts_max_heap            |

|                              |   |   |
|------------------------------|---|---|
|                              | Map Task Memory   | mapreduce_map_memory_mb                   |
|                              | Reduce Task CPU Virtual Cores   | mapreduce_reduce_cpu_vcores               |
|                              | Reduce Task Maximum Heap Size   | mapreduce_reduce_java_opts_max_heap       |
|                              | Reduce Task Memory  | mapreduce_reduce_memory_mb                |
|                              | ApplicationMaster Java Maximum Heap Size  | yarn_app_mapreduce_am_max_heap            |
|                              | ApplicationMaster Virtual CPU Cores   | yarn_app_mapreduce_am_resource_cpu_vcores |
|                              |   | yarn_app_mapreduce_am_resource_mb         |
|                              | ApplicationMaster Memory  |   |
| No<br>de<br>JobHistoryServer | System Group  | history_process_groupname                 |
|                              | System User   | history_process_username                  |
|                              | JobHistory Server Advanced Configuration Snippet (Safety Valve) for yarn-site.xml   | jobhistory_config_safety_valve            |
|                              | JobHistory Server Advanced Configuration Snippet (Safety Valve) for mapred-site.xml | jobhistory_mapred_safety_valve            |
|                              | JobHistory Server Logging Advanced Configuration Snippet (Safety Valve)             | log4j_safety_valve                        |
|                              | Java Configuration Options for JobHistory Server                                    | mr2_jobhistory_java_opts                  |
|                              | Heap Dump Directory   | oom_heap_dump_dir                         |
|                              | Dump Heap When Out of Memory  | oom_heap_dump_enabled                     |
|                              | Kill When Out of Memory   | oom_sigkill_enabled                       |
|                              | Automatically Restart Process   | process_auto_restart                      |
|                              | Job History Files Cleaner Interval  | mapreduce_jobhistory_cleaner_interval     |
|                              | Job History Files Maximum Age   | mapreduce_jobhistory_max_age_ms           |
|                              | MapReduce ApplicationMaster Staging Root Directory                                  | yarn_app_mapreduce_am_staging_dir         |
|                              | Java Heap Size of JobHistory Server in Bytes  | mr2_jobhistory_java_heapsize              |

|   |   |
|---|---|
| Cgroup CPU Shares   | rm_cpu_shares                                   |
| Cgroup I/O Weight   | rm_io_weight                                    |
| Cgroup Memory Hard Limit  | rm_memory_hard_limit                            |
| Cgroup Memory Soft Limit  | rm_memory_soft_limit                            |
| Hadoop Metrics2 Advanced Configuration Snippet (Safety Valve)                 | hadoop_metrics2_safety_valve                    |
| CGroups Hierarchy   | linux_container_executor_cgroups_hierarchy      |
| NodeManager Logging Advanced Configuration Snippet (Safety Valve)             | log4j_safety_valve                              |
| Healthchecker Script Arguments  | mapred_healthchecker_script_args                |
| Healthchecker Script Path   | mapred_healthchecker_script_path                |
| Java Configuration Options for NodeManager                                    | node_manager_java_opts                          |
| NodeManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml   | nodemanager_config_safety_valve                 |
| NodeManager Advanced Configuration Snippet (Safety Valve) for mapred-site.xml | nodemanager_mapred_safety_valve                 |
| Heap Dump Directory   | oom_heap_dump_dir                               |
| Dump Heap When Out of Memory  | oom_heap_dump_enabled                           |
| Kill When Out of Memory   | oom_sigkill_enabled                             |
| Automatically Restart Process   | process_auto_restart                            |
| Localized Dir Deletion Delay  | yarn_nodemanager_delete_debug_delay_sec         |
| Enable Shuffle Auxiliary Service  | mapreduce_aux_service                           |
| Containers Environment Variable   | yarn_nodemanager_admin_env                      |
| Container Manager Thread Count  | yarn_nodemanager_container_manager_thread_count |
| Cleanup Thread Count  | yarn_nodemanager_delete_thread_count            |
| Containers Environment Variables Whitelist                                    | yarn_nodemanager_env_whitelist                  |
| Heartbeat Interval  | yarn_nodemanager_heartbeat_interval_ms          |
| NodeManager Local Directory List  | yarn_nodemanager_local_dirs                     |

|                  |   |  |
|------------------|---|--|
|                  | Localizer Cache Cleanup Interval  | yarn_nodemanager_localizer_cache_cleanup_interval_ms |
|                  | Localizer Cache Target Size   | yarn_nodemanager_localizer_cache_target_size_mb      |
|                  | Localizer Client Thread Count   | yarn_nodemanager_localizer_client_thread_count       |
|                  | Localizer Fetch Thread Count  | yarn_nodemanager_localizer_fetch_thread_count        |
|                  | NodeManager Container Log Directories   | yarn_nodemanager_log_dirs                            |
|                  | Log Retain Duration   | yarn_nodemanager_log_retain_seconds                  |
|                  | Remote App Log Directory  | yarn_nodemanager_remote_app_log_dir                  |
|                  | Remote App Log Directory Suffix   | yarn_nodemanager_remote_app_log_dir_suffix           |
|                  | Java Heap Size of NodeManager in Bytes  | node_manager_java_heapsize                           |
|                  | Cgroup CPU Shares   | rm_cpu_shares  |
|                  | Cgroup I/O Weight   | rm_io_weight   |
|                  | Cgroup Memory Hard Limit  | rm_memory_hard_limit                                 |
|                  | Cgroup Memory Soft Limit  | rm_memory_soft_limit                                 |
|                  | Container Virtual CPU Cores   | yarn_nodemanager_resource_cpu_vcores                 |
|                  | Container Memory  | yarn_nodemanager_resource_memory_mb                  |
| Resource Manager | Hadoop Metrics2 Advanced Configuration Snippet (Safety Valve)                     | hadoop_metrics2_safety_valve                         |
|                  | ResourceManager Logging Advanced Configuration Snippet (Safety Valve)             | log4j_safety_valve                                   |
|                  | Heap Dump Directory   | oom_heap_dump_dir                                    |
|                  | Dump Heap When Out of Memory  | oom_heap_dump_enabled                                |
|                  | Kill When Out of Memory   | oom_sigkill_enabled                                  |
|                  | Automatically Restart Process   | process_auto_restart                                 |
|                  | Java Configuration Options for ResourceManager                                    | resource_manager_java_opts                           |
|                  | ResourceManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml   | resourcemanager_config_safety_valve                  |
|                  | ResourceManager Advanced Configuration Snippet (Safety Valve) for mapred-site.xml | resourcemanager_mapred_safety_valve                  |



|   |   |
|---|---|
| ResourceManager Advanced Configuration Snippet (Safety Valve) for nodes_allow.txt   | rm_hosts_allow_safety_valve                                 |
| ResourceManager Advanced Configuration Snippet (Safety Valve) for nodes_exclude.txt | rm_hosts_exclude_safety_valve                               |
| Capacity Scheduler Configuration  | resourcemanager_capacity_scheduler_configuration            |
| Fair Scheduler Assign Multiple Tasks  | resourcemanager_fair_scheduler_assign_multiple              |
| Fair Scheduler XML Advanced Configuration Snippet (Safety Valve)                    | resourcemanager_fair_scheduler_configuration                |
| Fair Scheduler Preemption   | resourcemanager_fair_scheduler_preemption                   |
| Fair Scheduler Size-Based Weight  | resourcemanager_fair_scheduler_size_based_weight            |
| Fair Scheduler User As Default Queue  | resourcemanager_fair_scheduler_user_as_default_queue        |
| ApplicationMaster Monitor Expiry  | yarn_am_liveness_monitor_expiry_interval_ms                 |
| NodeManager Monitor Expiry  | yarn_nm_liveness_monitor_expiry_interval_ms                 |
| Admin Client Thread Count   | yarn_resourcemanager_admin_client_thread_count              |
| ApplicationMaster Max Retries   | yarn_resourcemanager_am_max_retries                         |
| ApplicationMaster Monitor Interval  | yarn_resourcemanager_amliveliness_monitor_interval_ms       |
| Client Thread Count   | yarn_resourcemanager_client_thread_count                    |
| Container Monitor Interval  | yarn_resourcemanager_container_liveness_monitor_interval_ms |
| Max Completed Applications  | yarn_resourcemanager_max_completed_applications             |
| NodeManager Monitor Interval  | yarn_resourcemanager_nm_liveness_monitor_interval_ms        |
| Enable ResourceManager Recovery   | yarn_resourcemanager_recovery_enabled                       |
| Resource Tracker Thread Count   | yarn_resourcemanager_resource_tracker_client_thread_count   |
| Scheduler Class   | yarn_resourcemanager_scheduler_class                        |
| Scheduler Thread Count  | yarn_resourcemanager_scheduler_client_thread_count          |
| Java Heap Size of ResourceManager in Bytes  | resource_manager_java_heapsize                              |
| Fair Scheduler Node Locality Threshold  | resourcemanager_fair_scheduler_locality_threshold_node      |
| Fair Scheduler Rack Locality Threshold  | resourcemanager_fair_scheduler_locality_threshold_rack      |
| Cgroup CPU Shares   | rm_cpu_shares   |

|  |   |   |
|--|---|---|
|  | Cgroup I/O Weight                           | rm_io_weight                                      |
|  | Cgroup Memory Hard Limit                    | rm_memory_hard_limit                              |
|  | Cgroup Memory Soft Limit                    | rm_memory_soft_limit                              |
|  | Enable Fair Scheduler Continuous Scheduling | yarn_scheduler_fair_continuous_scheduling_enabled |
|  | Fair Scheduler Node Locality Delay          | yarn_scheduler_fair_locality_delay_node_ms        |
|  | Fair Scheduler Rack Locality Delay          | yarn_scheduler_fair_locality_delay_rack_ms        |
|  | Container Memory Increment                  | yarn_scheduler_increment_allocation_mb            |
|  | Container Virtual CPU Cores Increment       | yarn_scheduler_increment_allocation_vcores        |
|  | Container Memory Maximum                    | yarn_scheduler_maximum_allocation_mb              |
|  | Container Virtual CPU Cores Maximum         | yarn_scheduler_maximum_allocation_vcores          |
|  | Container Memory Minimum                    | yarn_scheduler_minimum_allocation_mb              |
|  | Container Virtual CPU Cores Minimum         | yarn_scheduler_minimum_allocation_vcores          |

### 6.3 Parameters for Spark service

| Roles          | Display Name   | API Name                                   |
|----------------|--|--|
| History Server | History Server Environment Advanced Configuration Snippet (Safety Valve) | SPARK_HISTORY_SERVER_role_env_safety_valve |
|                | History Server Logging Advanced Configuration Snippet (Safety Valve)     | log4j_safety_valve                         |
|                | Automatically Restart Process  | process_auto_restart                       |
|                | Java Heap Size of History Server in Bytes                                | history_server_max_heapsize                |
|                | Maximum Process File Descriptors   | rlimit_fds                                 |
|                | Cgroup CPU Shares  | rm_cpu_shares                              |
|                | Cgroup I/O Weight  | rm_io_weight                               |
|                | Cgroup Memory Hard Limit   | rm_memory_hard_limit                       |
|                | Cgroup Memory Soft Limit   | rm_memory_soft_limit                       |

|        |  |                                    |
|--------|--|------------------------------------|
| Master | Master Environment Advanced Configuration Snippet (Safety Valve) | SPARK_MASTER_role_env_safety_valve |
|        | Master Logging Advanced Configuration Snippet (Safety Valve)     | log4j_safety_valve                 |
|        | Automatically Restart Process                                    | process_auto_restart               |
|        | Additional Master args   | master_additional_args             |
|        | Java Heap Size of Master in Bytes                                | master_max_heapsize                |
|        | Maximum Process File Descriptors                                 | rlimit_fds                         |
|        | Cgroup CPU Shares  | rm_cpu_shares                      |
|        | Cgroup I/O Weight  | rm_io_weight                       |
|        | Cgroup Memory Hard Limit   | rm_memory_hard_limit               |
|        | Cgroup Memory Soft Limit   | rm_memory_soft_limit               |
| Worker | Worker Environment Advanced Configuration Snippet (Safety Valve) | SPARK_WORKER_role_env_safety_valve |
|        | Worker Logging Advanced Configuration Snippet (Safety Valve)     | log4j_safety_valve                 |
|        | Automatically Restart Process                                    | process_auto_restart               |
|        | Total Java Heap Sizes of Worker's Executors in Bytes             | executor_total_max_heapsize        |
|        | Work directory   | work_directory                     |
|        | Additional Worker args   | worker_additional_args             |
|        | Java Heap Size of Worker in Bytes                                | worker_max_heapsize                |
|        | Maximum Process File Descriptors                                 | rlimit_fds                         |
|        | Cgroup CPU Shares  | rm_cpu_shares                      |
|        | Cgroup I/O Weight  | rm_io_weight                       |
|        | Cgroup Memory Hard Limit   | rm_memory_hard_limit               |
|        | Cgroup Memory Soft Limit   | rm_memory_soft_limit               |