Mosquito Benchmark Framework Manual

September 9, 2014

Contents

1 Introduction

The Mosquito Benchmark Framework (MBF) is both a program and an API framework for benchmarking SPARQL Endpoints.
On one side it provides the ability to benchmark different SPARQL endpoints (e.g. triplestores) with given or generated datasets and predefined test cases and on the other side it provides a framework and API to implement either a complete different benchmark or just parts of the benchmark (e.g. own test cases).

In the following the manual will describe first how to use MBF as a framework then how to use it as a benchmark and how to configure it.
In the last part there will be a list of further information (such as where to find the source code)

2 Framework/API

2.1 Setting up

To use MBF as an API for your own benchmark, please consider the javadoc.

To use MBF as a framework you don't have to change the code itself, just write a library (jar file) and use MBF as a library for your code, which is necessary to include. If you deploy your code as a library for MBF you don't need MBF as a library in your deployed code.
MBF will load your classes on runtime if they are in the lib folder by the MBF jar file.

2.2 Interfaces

To replace parts of the Benchmark with your code MBF use Class.forName(className) to load your classes on runtime. To achieve this and also give you the possibility for properties, the classes must implement the provided interfaces. There are three parts you can change like this and therefore three interfaces.
Write a class which implements one of these interfaces and you can use it by writing the class name into the configuration file (see also 3.4).

2.2.1 Testcase

To use your testcases in MBF you have to implement the interface *Testcase.java*. Therefore you have to implement the following functions:

Please be aware that *ResultSet* is a class provided by MBF (see also 2.2.1.1) and *Connection* is a class provided by an adjusted version of wookieConnection (see also 2.2.1.2 and 4.1)

Function	**parameters**	**return**	**Description**
start	none	void	This starts the whole testcase after setting the properties, current connection, dbname and percentage
getResults	none	Collection<ResultSet>	This will return the results after starting the testcase
addCurrentResults	Collection<ResultSet>	void	This will add the previous results for the specific percentage. Which means you'll get the results for all tested connections for the current percentage.
setProperties	Properties	void	If you want to give users a little bit more control over your testcase, you can use properties for parameters etc.
setConnection	Connection	void	This will set the connection to test
setCurrentDBName	String	void	This will give you the current id of the tested connection
setCurrentPercent	String	void	This will set the current percentage on which the connection will be tested

2.2.1.1 ResultSet

The *ResultSet* is an Iterator which gives you the opportunity to rapidly go through the current results and easily add results to it. Also you can save the *ResultSet* as a CSV file (the *ResultSet* ist simply a table with a header) and as a bar chart png file.
Example:

header:	Connection	Query1	Query2	...	QueryN
Table:	con1	200	230	...	176
	con2	400	120	...	300
	⋮	⋮	⋮	⋮	⋮
	conM	234	1233	...	89

If this is the header and table in your *ResultSet* and you have one more connection to test, you can simply add the row

conM1	123	123	...	123

by the function *addRow*.
For more information on the *ResultSet* please consider the javadoc.

2.2.1.2 Connection

The connection has several functions to give you an interface for SPARQL endpoints which is easy to use.
The connection can...

- ...upload a file either in the default graph or in a given graph.
- ...execute a SPARQL/UPDATE query (as String)
- ...execute a SPARQL query (as String) and get a *java.sql.ResultSet*
- ...execute an UPDATE query (as String) and get a boolean if it worked.
- ...set the upload type to POST or PUT
- ...drop a given graph

2.2.2 Clusterer

To use your own log clustering in MBF you have to implement the interface *Clusterer.java*. Therefore you have to implement the following functions:

Function	**parameters**	**return**	**Description**
cluster	String logPath String queriesFile	void	This is the main method of your clusterer. You'll get the path of the log files and the name of the output file (queriesFile) where the queries should be written in.
setProperties	Properties	void	If you want to give users a little bit more control over your log clustering process, you can use properties for parameters etc.

2.2.3 ConverterI with RDFVocabulary

To convert your data from non RDF files into RDF Files (RDF/XML, TURTLE, N-TRIPLE,...) you can write your own Converter simply by implementing the interface *ConverterI* and write a class which extends the class *RDFVocabulary*. The first one will simply convert the files while the latter one just setup the namespace etc. the converter should use. The *RDFVocabulary* will be initialized before the converter starts. Be aware that even though the *ConverterI* only needs the function to convert one file, the rest will be handled by the benchmark class itself.

ConverterI

Function	parameters	return	Description
processToStream	String inputFile String outputFile	Boolean	This is the main method of the converter and will convert the given input file to the specified output file.
setOutputFormat	String outputFormat	void	This will simply just set the outputFormat. It will be set before the converting process

RDFVocabulary

Function	parameters	return	Description
init	String namespace String anchor String prefix String resourceURI String propertyPrefixName String resourcePrefixName	void	This will just set the user wanted resourceURI etc. to the converter.

3 Benchmark

3.1 In general

The MBF as a benchmark can . . .

- . . . convert non RDF data into a RDF dataset and use this as the 100% dataset
- . . . cluster and analyze query logs with different strategies and writes out query patterns to test
- . . . generate smaller and greater datasets out of an initial 100% dataset.
- . . . upload the datasets before testing them and measure the time if the UploadTestcase is specified in the configuration
- . . . test different testcases on different connections (SPARQL endpoints) with different sized datasets
- . . . save all results as csv files and if possible save them as barchart pgn files.
- . . . send you an email if the benchmark succeeded with the results attached (in a zip folder) or aborted with the exception that causes the abortion and the by then calculated results.

3.2 Starting the benchmark

Before you start you must install Mosquito.

1. Download the latest version in a choosen folder and all the dependent libraries in a folder lib in the choosen folder.
2. If you want to use the data generation part of the benchmark with the specific type "coherence" you'll need lp_solve 5.5.2.0. You can download it here : http://sourceforge.net/projects/lpsolve/files/lpsolve/5.5.2.0/ otherwise skip to 3.
    (a) The files you need are platform specfic (replace "platform" with your platform. Be aware that you can have a x64 processor but your JVM is a 32-bit installation. If this is the case you need the 32-bit files):
        i. lp_solve_5.5.2.0_exe_"platform" .zip or .tar.gz
        ii. lp_solve_5.5.2.0_dev_"platform" .zip or .tar.gz
    (b) Extract the libraries (or if you're lazy all the files) into a new folder called lpsolve_lib in your folder with the mosquito.jar file
3. configure your config.xml file as you wish

To start the benchmark open a terminal and write the following:

```
>>java -cp "mosquito.jar;lib/*"
       de.uni_leipzig.mosquito.benchmark.Benchmark config.xml
```

if you're using the data generator with the "coherence" type you'll need to start the benchmark as following:

```
>>java -Djava.library.path=lpsolve_lib -cp "mosquito.jar;lib/*"
       de.uni_leipzig.mosquito.benchmark.Benchmark config.xml
```

Where config.xml is the configuration xml file you use (see also 3.4)

3.3 Implementations

As MBF has several parts which are exchangeable and provides the interfaces for this case there are some implementations yet.

3.3.1 PGNConverter

This converter simply converts PGN-files into rdf data.
You can use it with the following two class-names:
converter-class: de.uni_leipzig.informatik.swp13_sc.converter.PGNToRDFConverterRanged
rdf-vocabulary-class: de.uni_leipzig.informatik.swp13_sc.datamodel.rdf.ChessRDFVocabulary

3.3.2 SortedStructureClusterer

The SortedStructureClusterer clusters the queries in the log files through their structures and returns the frequentest queries for the frequentest structures.
The following properties can be set:

name	**values/type**	**description**	**required?**
threshold-queries	Integer	The minimum number a query must occur in the log files to be part of the clustering	default=10
threshold-structs	Integer	The minimum number a structure must occur in the log files to be part of the clustering	default=10

class: de.uni_leipzig.mosquito.clustering.clusterer.SortedStructureClusterer

3.3.3 BorderFlowQueryClusterer

The BorderFlowQueryClusterer is a log cluster and analyzing implementation [3] using the Border Flow library (see also 4.1)

Even though it works it is not fully implemented in the current version!
The following properties can be set:

name	**values/type**	**description**	**required?**
threshold-queries	Integer	The minimum number a query must occur in the log files to be part of the clustering	default=10
delta	Integer	Parameter for threshold calculation [3]	default=2
min-nodes	Integer	The minimum nodes a cluster must have to be part of the final query selection	default=3
conn-threshold	Double [0:1]	Threshold parameter for Border Flow algorithm (see also 4.1)	default=0.8
test-one	Boolean	Parameter for Border Flow algorithm (see also 4.1)	default=true
heuristic	Boolean	Parameter for Border Flow algorithm (see also 4.1)	default=true
caching	Boolean	Parameter for Border Flow algorithm (see also 4.1)	default=true
harden	hardensuperset hardensharedshed	Parameter for Border Flow algorithm (see also 4.1)	required
quality	qualitymeasurerelativeflow qualitymeasuresilhoutte	Parameter for Border Flow algorithm (see also 4.1)	required

class: de.uni_leipzig.mosquito.clustering.clusterer.BorderFlowQueryClusterer

3.3.4 BorderFlowStructureClusterer

Same as the BorderFlowQueryClusterer but instead clustering the Queries with their similarities the Structures will be clustered.
class: de.uni_leipzig.mosquito.clustering.clusterer.BorderFlowStructureClusterer

3.3.5 Testcases

3.3.5.1 UploadTestcase This testcase is needed if you need to upload datasets to your connections!
It simply uploads the files for the datasets and measure the time it needs to take to upload the current file.
class: de.uni_leipzig.mosquito.testcases.UploadTestcase

3.3.5.2 QueryTestcase The query testcase will test the query patterns for a given time-limit against one Connection.
The query testcase can handle a query pattern file with SPARQL queries and/or UPDATE queries in it or a query pattern file with SPARQL queries and you can set a path where your changesets are located. Changesets must have the following regex form: [0-9]{6}(added—removed).nt
Added means they will be inserted, removed means the will be deleted through the testcase.
The query testcase needs a file where the query patterns are written in (one query per one line). These query patterns can contain variables like %%var%% or %%var2%% etc. These variables will be turned into real data results described in the following lines.
The first step the query testcase will do (if the benchmark hasn't done it yet) is to convert the query patterns into queries. To achieve this the patterns will be converted to select queries with the variables of the pattern as SELECT variables. For example let the following query be the third query of the query patterns file:

```
SELECT ?v1 ?v2 WHERE {%%var1%% <URI> ?v1. ?v1 ?v2 "2"}
```

As you can see var1 is a pattern variable and to test the connection with queries which have results, the pattern will be converted into the following query

```
SELECT ?var1 WHERE {?var1 <URI> ?v1. ?v1 ?v2 "2"}
```

Now all results (maximum the given limit) will be saved in the file 3.txt, by replacing %%var1%% with the results, in the directory QueryTestcase.
This will happen for every query pattern in the query patterns file.
The second step is to test all queries in a given order against the connection.
The given order can be set with the updateStrategy (In the following the UPDATE query can be also changesets if you're using changesets instead of UPDATE queries).
if you set this to fixed you can either set a parameter x or it will be calculated. This states that after every SPARQL (UPDATE) query x UPDATE (SPARQL) queries be tested (It'll be calculated if there are more UPDATE or more SPARQL queries).
if you set this to variation you can either set the parameter x or it will be calculated. Herby the x sets an interval like the following [1:x] if set or it will

calculate an accurate interval.
After every run of SPARQL (UPDATE) and the following UPDATE (SPARQL) the new x will be calculated (a random number in the given interval)
If you're using live data (changesets) instead of UPDATE queries you can set the ldLinking parameter.
This parameter states in what order to execute the changesets.

strategy	**resulting order**
insertsFirst	This states the every 'added' changeset (orderd by their number) will be before the 'removed' changesets (orderd by their number). So you have 000010.added.nt before 000234.added.nt and 999999.added.nt before 000001.removed.nt and so on.
deletesFirst	Like insertsFirst but first 'removed' then 'added' changesets
ID	This states that the changesets will be first ordered by the number and then by 'added' before 'removed'. So you have 000011.added.nt before 000011.removed.nt but directly after 000010.removed.nt ...
DI	Like ID but 'removed' changesets before 'added' ones.

The following properties can be set:

name	**values/type**	**description**	**required?**
queryPatternFile	String	The filename with the query patterns	required
updateStrategy	fixed variation	The described strategy to use if you're using SPARQL and either UPDATE queries or live data files	optional
x	Integer	If updateStrategy is set this is the parameter which specified the described parameter x	optional
time-limit	Integer	The time in milliseconds how long the testcase should run	default=3600000
limit	Integer	The number of how much queries should be generated for one query pattern	default=5000
ldLinking	insertsFirst deletesFirst ID DI	The internal live data strategy to consider which is the next changeset to querying	(only with ldPath)
ldPath	String	If you're using live data you can set the path where your changesets are located	(only with ldLinking)

class: de.uni_leipzig.mosquito.testcases.QueryTestcase

3.3.5.3 StressTestcase The stress testcase provides an approach to test a real live situation with a given number of users.
It will test the QueryTestcase with several threads (users) to achieve this real live situation.
It takes the same properties like QueryTestcase (which are the properties the threads will have) plus the property *user* and its value.
class: de.uni_leipzig.mosquito.testcases.StressTestcase

3.4 Configuration

The MBF is configured through a XML file.

3.4.1 Root node

The root node is required and every section after the root node section is a tag inside this root node.
The root node looks like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<mosquito xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      ...
</mosquito>
```

3.4.2 Data description

The data description tag contains the description how the converter will handle the data. For example which resourceURI the converter should use.
The data-description tag itself is only required if a converter will be used.
The following table shows the tags the data-description needs

tagname	attributes	description	required?
namespace	name=URI	This provides the actual namespace	required
anchor	name=CDATA	This provides the property suffix so the property URI will be namespace+anchor	required
resource-uri	name=URI	This provides the resource-uri	required
property-prefix-name	name=CDATA	this is needed if you use outputFormat like TURTLE so the propertyURIs can be replaced by the prefix	required
resource-prefix-name	name=CDATA	this is needed if you use outputFormat like TURTLE so the resourceURIs can be replaced by the prefix	required

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<mosquito xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <data-description>
              <namespace name="http://localhost/" />
              <anchor name="property/" />
              <resource-uri name="http://localhost/resource/" />
              <property-prefix-name name="localProp" />
              <resource-prefix-name name="lokalRes" />
       </data-description>
       ...
</mosquito>
```

3.4.3 Databases

The databases tag contains several database tags.

The database tag has two attributes, one is the id tag and the other one is the type tag. The type can be either *impl* or *curl*.
The impl type converts files which will be uploaded into an INSERT tag. Also all UPDATE queries will be send directly over the SPARQL endpoint. This can be slow and the RAM can be very quickly out of memory.
The curl type sends all UPDATE queries through a specified script command (e.g. curl). The tags which will be only for the curl type have the prefix *curl-*. On these one you can use several variables like $USER, $PWD, $CURL-URL, $GRAPH-URI, $UPLOAD-TYPE, $FILE, $UPDATE
Some of them can be replaced by the explicit string, but the variables FILE, UPDATE and GRAPH-URI must be set.
Both use the SPARQL endpoint directly for SPARQL queries.

tagname	**attributes**	**description**	**required?**
endpoint	uri	The SPARQL endpoint	required
user	value	The user for the endpoint	optional
pwd	value	The password for the user	optional
curl-url	url	The curl-url (as it can differs from the endpoint)	(only curl)
curl-command	command	The script command to upload a file	(only curl)
curl-drop	command	The script command to drop a specific graph	(only curl)
curl-update	command	The script command to send an UPDATE query	(only curl)

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<mosquito xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ...
        <databases>
                <database id="curl" type="curl">
                        <endpoint uri="localhost/sparql-auth" />
                        <user value="dba" />
                        <pwd value="dba" />
                        <curl-url url="http://localhost/sparql-graph-crud-auth" />
                        <curl-command command="curl --digest --user
                                $USER:$PWD --url '$CURL-URL?graph-uri=$GRAPH-URI'
                                $UPLOAD-TYPE -T $FILE"/>
                        <curl-drop command="curl -X DELETE
                                $GRAPH-URI --url '$CURL-URL'"/>
                        <curl-update command="curl -i -d '$UPDATE' -u '
                                $USER:$PWD' -H 'Content-Type:
                                application/sparql-query' --url 'http://$ENDPOINT'"/>
                </database>
                <database id="impl" type="impl">
                        <endpoint uri="localhost/sparql-auth" />
                        <user value="dba" />
                        <pwd value="dba" />
                </database>
                <database id="impl2" type="impl">
                        <endpoint uri="localhost/sparql" />
                </database>
        </databases>
        ...
</mosquito>
```

3.4.4 Log-clustering

The log-clustering tag which is optional, contains three attributes which are all required and are the following:

attribute	description
class	The full classname. This states the class to use for the log clustering process. Remember this one must be a class which implements the Clusterer.java interface
path	The path of the logfiles
output-file	The output file in which the final queries should be written in.

The output-file can be used in the testcases. So you can both test clustered queries and some queries you provide if you want to.
To set a property simply add a tag inside the log-clustering tag, called property, with the attributes name and value. Name declares the property name and value the value for the property. These properties will be set to the *Clusterer* as a *Properties* object.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<mosquito xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ...
    <log-clustering
        class="de.uni_leipzig.mosquito.clustering.clusterer.BorderFlowQueryClusterer"
        path="../../LogFiles/"
        output-file="Queries.txt">

        <property name="delta" value="4"/>
        <property name="test-one" value="true"/>
    </log-clustering>
        ...
</mosquito>
```

3.4.5 Benchmark

To convert your files into RDF files (notice that these files will be string together to one file which will be used as the initial file and will be uploaded in the reference connection) before the benchmark starts you must add the following five tags.

tag name	**attributes**	**description**
convert-processing	value	The tag to state if there is data which must be converted
convert-input-path	name	The path of the files which should be converted
convert-class	class	The actual Converter class name to use for converting the files
rdf-vocabulary-class	class	The class name to use for the RDFVocabulary.
output-path	name	This sets the path in which the converted files will be saved in (Be aware that you don't need the output-path or the file-names in it to test the dataset and/or generated new datasets out of it. Just let the initFile attribute of the random-function tag empty. The benchmark will handle it.)

If you want to drop the connection (or if graph-uri is set the graph) every time the connection will be tested (for every percent) you can set the **drop-db** tag with the attribute *value* which can be true, to state that the dataset in the connection should be dropped, or false, to state that the dataset should not be dropped.
Be aware that if you specify a *graph-uri* tag it will only drop this specific graph.
It is highly recommend to set this tag to true!

To set that the tests should only be tested on a specific graph you can add a **graph-uri** tag which has the attribute *name*. In this tag you simply write the graph URI which will be tested.

If you use one of the implemented testcases or some other which uses the *Query-Handler* class to convert the query patterns into queries, you can set how many queries will be generated for one pattern by adding a **query-diversity** tag. This tag has one attribute called *value*, which will set the max. integer how many queries to generate for one pattern.

The **random-function** tag states which percentage/files will be tested and if they needs to be generated.
The tag itself has three attributes. The first one is called *generate* which states if the datasets (smaller and greater than 100%) must be generated.
To set the algorithm for generating smaller datasets there is the second attribute which is called *type*. This attribute can be either *RandomInstance*,

RandomTriple or *coherence*. The first two are the generation algorithms out of the first DBpedia SPARQL benchmark paper[2]. In this paper they are called *seed* and *rand*. The latter one is an implementation based on coherence of the dataset [1]. The third attribute sets the initial file (the 100% file). This should be empty if the converter generates this file.
If you use the type *coherence* you can specify two more attributes by adding *roh* and *coherence* as attributes. roh is the relaxation parameter specified in the referenced paper and coherence is the parameter to maintain by the generation which is also specified in the referenced paper.
To set which percentages should be tested you can add a **percent** tag in the *random-function* tag. The *percent* tag has two attributes. The first one called *value* which should be a double (1.0 stands for 100%) and states the percentage to test. The second attribute is called *file-name* which can set the name of the file if you don't need to generate it and you have the specific file.

To warmup the connections before you test them (this will happen for every percentage you'll set) you can add a **warmup** tag which has two attributes. The first one is the *time* attribute. The value is the time in milliseconds how long the warmup will go on. The second one is the *file-name* attribute. This specify where to find the file with the queries (not query patterns!) to warmup the connection with.

The **test-db** tag sets which databases (must be existing ones in the databases tag) will be tested.
The test-db tag has two attributes, the first one is the *reference* attribute which is the id (the database tag id) which of the given connections should be used as a reference (for example to convert the query patterns into queries with real data in it).
The second attribute is the *type* attribute which can be either *all* or *choose*. If it's set to *all* every connection in the databases tag will be tested except for the reference connection. If it's set to *choose* only the connection specified in the *test-db* tag will be tested (even though you can specify the reference connection to be tested it will not be tested!).
To specify the connections you add a *db* tag in the *test-db* tag with the attribute *id*. The id attribute must match one of the database ids.

3.4.5.1 Testcases The testcases tag contains several testcase tags. The testcases tag is required even though it can be empty.

The testcase tag has one attribute called *class* in which the full class name of the testcase class should stand. Remember that the testcase class must implement the *Testcase.java* interface.
To set a property simply add a tag inside the testcase tag called property, with the attributes name and value. Name declares the property name and value the value for the property. These properties will be set to the *Clusterer* as a

Properties object.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<mosquito xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ...
    <benchmark log="Benchmark">
        <convert-processing value="true"/>
        <convert-input-path name="src/test/resources/" />
        <rdf-vocabulary-class
          class="de.uni_leipzig.informatik.swp13_sc.datamodel.rdf.ChessRDFVocabulary"/>
        <converter-class
          class="de.uni_leipzig.informatik.swp13_sc.converter.PGNToRDFConverterRanged"/>
        <output-path name="src/test/java/"/>
        <drop-db value="false" />
        <graph-uri name="http://example.com" />
        <query-diversity value="1" />
        <random-function type="coherence" generate="true" initFile="" coherence="0.75" roh="
                <percent value="0.5" file-name=""/>
                <percent value="0.25" file-name=""/>
        </random-function>
        <warmup time="20000" file-name="Q.txt"/>
        <test-db type="choose" reference="impl2">
                <db id="curl" />
                <db id="impl" />
        </test-db>
        <testcases>
                <testcase class="de.uni_leipzig.mosquito.testcases.QueryTestcase">
                        <property name="queryPatternFile" value="Q.txt"/>
                        <property name="limit" value="2"/>
                        <property name="time-limit" value="3600000"/>
                        <property name="updateStrategy" value="fixed"/>
                </testcase>
                <testcase class="de.uni_leipzig.mosquito.testcases.StressTestcase">
                        <property name="users" value="3"/>
                        <property name="queryPatternFile" value="Q.txt"/>
                </testcase>
                <testcase class="de.uni_leipzig.mosquito.testcases.UploadTestcase"/>
        </testcases>
    </benchmark>
        ...
</mosquito>
```

3.4.6 Email notification

The optional email-notification tag has one attribute called attach-results and can be set either true or false. If this attribute is set true, the results (or temporary results) will be zipped and attached to the email.

tagname	attributes	description	required?
hostname	CDATA	The hostname of the email server	required
port	Integer	The port of the host	required
user	CDATA	The user from which the email will be send	required
password	CDATA	The password for the user. For security reason you can simply leave this tag away and the benchmark will ask you over the command line.	optional
email-name	CDATA	The adress which will be shown in the sended email	required
email-to	CDATA	The adresses to send the email to. Be aware if you want to send it to the email-name too you must add this explicit. Can be set multiply times	required

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<mosquito xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ...
    <email-notification attach-results="false">
        <hostname value="smtp.gmail.com"/>
        <port value="465"/>
        <user value="what@ever.com"/>
        <password value="secret"/>
        <email-name address="whatever@youwant.com"/>
        <email-to address="what@ever.com"/>
        <email-to address="another@one.com"/>
    </email-notification>
</mosquito>
```

4 Further information

4.1 Links

Beware that these following information can be outdated

website	`https://aksw.github.io/mosquito/`
distribution	`https://github.com/AKSW/mosquito/tree/master`
javadoc	`https://aksw.github.io/mosquito/javadoc`
source code	`https://github.com/AKSW/mosquito`
issue tracker	`https://github.com/AKSW/mosquito/issues`
contact information	mai12cpg@studserv.uni-leipzig.de
Border Flow (Clustering algorithm)	`http://borderflow.sourceforge.net`
DBpedia SPARQL Benchmark	`http://aksw.org/Projects/DBPSB.html`
adjusted wookieConnection	`https://github.com/AKSW/mosquito/blob/develop/de.uni_leipzig.mosquito/src/main/resources/lib/connection-0.0.1-SNAPSHOT.jar`

4.2 License

MBF itself is licensed under the MIT License [A], but be aware that there are libraries we use which are licensed under different licenses.

References

[1] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and oranges: A comparison of rdf benchmarks and real rdf datasets. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 145–156, New York, NY, USA, 2011. ACM.

[2] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data. In *ISWC 2011*.

[3] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. Usage-Centric Benchmarking of RDF Triple Stores. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*, 2012.

A The MIT License (MIT)