FP7 Project ASAP
Adaptable Scalable Analytics Platform

ASAP D7.2
Integration Prototype "ASAP System Prototype v.2"

WP 7 – Integration of the ASAP System

Nature: Report

Dissemination: Public

Version History

Version	Date	Author	Comments
0.1	10 Feb 2016	Papagiannaki S.	Initial Version
0.2	25 Feb 2016	Papagiannaki S.	Revised Version
0.5	01 Jun 2016	Papagiannaki S., Pratikakis P., Chalkiadaki M.	Revised Version
1.0	11 Jun 2016	Papagiannaki S., Pratikakis P., Chalkiadaki M.	Final Version

Acknowledgment This project has received funding from the European Union's 7th Framework Programme for research, technological development and demonstration under grant agreement number 619706.

Contents

Chapter 1

Introduction

ASAP focuses on (i) innovative methods and technologies and (ii) tools and applications. Regarding methods and technology, we develop novel methods in order to model cost and performance of multiple data stores and analytics execution engines. Building on these, we perform automated job scheduling to multiple runtime and data store technologies together with real-time tracking of intermediate results. To deliver this technology to the end user, we couple it with state-of-the-art visualization tools enabling both qualitative and quantitative monitoring of a job's performance and cost. The integrated technology enables fast, easy development and submission of both simple and highly complex analytics tasks that take full advantage of the existing resources according to user requirements. Overall, ASAP delivers open source tools that can be used both separately and as an integrated system in order to provide efficient execution and management of complex analytics tasks. This deliverable reports on the current status of work for the integration of all components into an integrated system.

Specifically, the main objectives of this work package are:

- Ensure the integration of the contributions in WP2, WP3, WP4, WP5 and WP6.
- Coordinate development and delivery of the integrated modules based on the research and development results in the different Research Areas.
- Ensure that these prototypes are used to integrate and coordinate the coherent delivery of the ASAP system for its application in WP8 and WP9.

1.1 Task Description

This deliverable reports on work done within tasks T7.2 and T7.3.

Task T7.2 integrates the technologies of the components from WP2, WP3, WP4, WP5 and WP6, based on the overall architecture defined in D1.2, in order to execute a subset of the use cases proposed in WP8 and WP9.

Task T7.3 tests alpha- and beta-versions of the integrated platform and the individual components. It gives continuous feedback to the application development and drives modifications to the integrated tool, as well as to the individual components and services. For this reason this Task continues until the end of the project. During task T7.2 we tested the functionality of the simple use cases developed in WP1 to verify

the correctness of the integrated system. We also tested the queries developed on the two applications using simple or reduced, heterogeneous data stores. All of these tests are automated. We test and verify the correct function of the integrated platform.

Chapter 2

ASAP Components

This chapter briefly describes the separate components of the ASAP system, depicted, along with their interactions, in Figure 2.1. Readers already familiar with every module, as reported in detail in the corresponding deliverables, can skip this chapter.

2.1 Intelligent, Multi-Engine Resource Scheduler (IReS)

The IReS platform, thoroughly described and implemented in the scope of WP3, targets the workflow optimization, examining alternative execution paths using various underlying engine and operator implementations. Using its web interface the user can define operators and datasets along with their properties and restrictions and store them in a language understandable by the other ASAP components and specified in WP5. Furthermore, it provides functionality for validating and executing workflows by extending the Apache Kitten7 framework [4] in order to execute over YARN [3], apart from separate operators, also workflows as a DAG of operators.

IReS is an open source[1] web application that exposes its functionality to the rest of the ASAP components through a RESTful API. It is implemented in Java using the Jetty [11] servlet engine and the Jersey [10] RESTful Web service framework.

2.2 Workflow Management Tool (WMT)

The WMT, described and implemented in the scope of WP5, provides full functionality for designing, editing, analyzing and optimizing an abstract workflow. It interacts with IReS for loading the registered operators and executing workflows via the RESTful API the latter provides. Its Analysis and Optimization functionality can be invoked by web actions that call the respective Python methods.

The WMT is an open source[2] Javascript [8] Web application rendered behind a Nginx [13] web server.

[1] `https://github.com/project-asap/IReS-Platform`

[2] `https://github.com/project-asap/workflow`

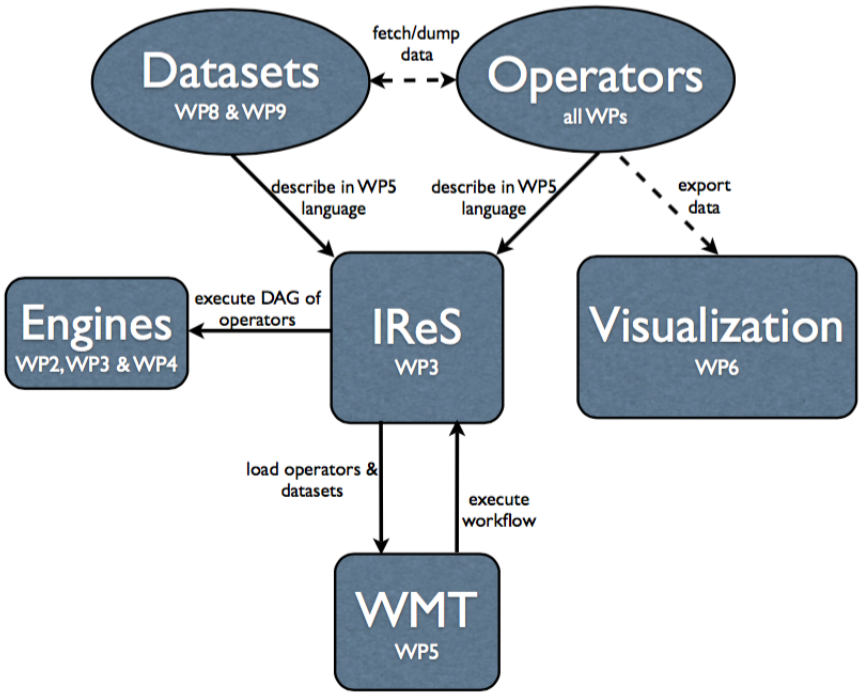

Figure 2.1: ASAP components

2.3 Asap Operators

In the scope of WP3 a number of popular analytics operations (TF/IDF, K-Means, Word2Vec etc) are modeled and profiled in several runtimes (Hadoop [1], Weka [19], Mahout [12]). In addition to this for the needs of the ASAP the following additional operators have been implemented, registered and profiled by IReS:

- K-Means, TF/IDF and Word2Vec implementations for Swan, developed in the scope of WP2.
- Web analytics operation implementations, developed in the scope of WP8.
- Peak detection and Sociometer implementations in Spark and Spark-Nested, developed in the scope of WP9.
- Operators for dumping data in the visualization dashboard, developed in the scope of WP9.

2.4 Engines

For the execution of the operators listed in the previous section, ASAP uses Hadoop [1] and more precilly HDFS [2], Mahout [12] and Weka [19]. In addition, we use Swan [21], an experimental extension of Cilk [20] for operators written to use a data-flow style of execution. Finally, we use the Spark-Nested frame-

work developed in WP4 for support of Spark applications that require nested transformations, hierarchical data representation and distributed scheduling.

2.5 Visualization Component

The visualization component, subject of the WP6, consists of:

- The ASAP dashboard which collects, queries and visualizes data. The dashboard is implemented using the D3 JavaScript library and it is hosted on a webLyzard server.
- Open APIs for ingesting data in an Elasticsearch [6] installation. The Document API is used for ingesting crawled Web documents (unstructured data) from WP8. The Statistical Data API is used for ingesting the telecommunications data (structured data) produced by WP9.

Chapter 3

ASAP Integration Summary

The life-cycle execution of a data analytics job in the scope of ASAP is summarised in the following steps:

1. The Developer designs a primitive computation as an abstractor operator using the IReS web interface.
2. The operator metadata, describing its semantics in the workflow description language developed in WP5, are stored in the ASAP operator library that resides in the IReS.
3. The Developer adds a number of data sources using the IReS web interface.
4. The data source metadata, describing its location in the workflow language developed in WP5, are stored in the ASAP operator library that resides in the IReS.
5. The Developer creates one or multiple implementations of the above operator, following the programming model proposed in WP2.
6. The Developer using the IReS web interface, stores the above implementations in the ASAP operator library as materialized operators.
7. The Developer updates the metadata describing the materialized operator's semantics in the workflow description language developed in WP5 by introducing scripts for its execution automation.
8. The IReS profiler builds a cost model of the operator implementations and saves them along with the materialized operator's metadata.
9. The WMT, during the initialization process, loads the ASAP library by making the respective request to the RESTful API of the IReS platformm.
10. The Workflow Designer, using the WMT web interface, designs a workflow by combining the available data sources and operators.
11. The Workflow Designer, using the WMT web interface, can analyze and optimize the workflow.
12. The Workflow Desinger, can save the workflow in the workflow language developed in WP5.

13. The User, using the WMT web interface, can load a existing workflow and initiate the execution of the computation by making the respective request to the API of the IReS platform.

14. The IReS platform schedules the workflow using the best possible execution plan, based on the operator metadata and costs.

15. The IReS platform orchestrates the execution of the selected execution plan by employing YARN for integrating with the various computing engines that lay underneath (e.g. Swan or Spark Nested documented in WP4).

16. An operator can dump results to the Elasticsearch installation of the Visualization component using the RESTful API introduced in WP6.

17. The User can see the intermediate or final computation results using the Visualization Dashboard described in WP6.

Chapter 4

Prototype Setup

This chapter describes the current status of the ongoing integration. Spcecifically, we describe the organization of code in repositories and provide brief directions on how one can deploy the corresponding modules and the system as a whole. For more detailed and up-to-date user guide and installation instructions, we refer the reader to the documentation within the repositories described below.

4.1 ASAP source code

The majority of the ASAP components are open source. Therefore a separate repository for each component has been created under the project's account in Github or it has been forked from a repository residing under another account. In each separate repository the collaborator Github accounts, which are authorised to push to the respective repository, are assigned. The table 4.1 shows the respective Github repositories.

4.2 Unified Setup using Fabric

As mentioned in the previous chapter, the ASAP system consists of a number of components developed using different technologies. In order to simplify the installation of all the components and their continuous and smooth integration we have employed Fabric [7]: a Python library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks. Fabric tasks are typically methods that execute shell commands easily and handle failures nicely.

In that way we have created such fabric tasks for each ASAP component. The typical scenario of such a task is more or less the following:

- install prerequisites
- get a copy of the component: e.g. clone the respective github repository.
- configure: create and modify configuration files.
- build: generate the executables (if necessary)
- test the base functionality

Repository	Description	Work Package	Collaborators
asap_operators forked from hvdieren/asap_operators	Swan Analytics Operators	WP2	hvdieren (QUB), Murphky(QUB)
IReS-Platform	IReS	WP3	npapa (ICCS), gsvic (ICCS), vpapaioannou (ICCS), cmantas (ICCS), kdoka (ICCS), polyvios (FORTH)
Spark-Nested	Spark Nested	WP4	polyvios (FORTH), papagian (FORTH), mhalkiad (FORTH), p01K (FORTH)
workflow	WMT	WP5	maxfil (GENEVE)
weblyzard_api forked from weblyzard's repository	Visualization Web Services	WP6	
fabric-scripts	Fabric scripts	WP7	papagian (FORTH)
web-analytics	Web analytics application	WP8	thanh-im (IMR), rigaux (IMR)
telecom-analytics	Telecom analytics application	WP9	papagian (FORTH), mhalkiad (FORTH), mesosbrodleto (UNIPI)

Table 4.1: Github repositories

The `bootstrap_wmt()` method in Appendix A is an example of such a bootstrap scenario for the WMT component.

Fabric scripts for the all the components developed in the scope of the ASAP, as well as third-party software employed by ASAP, such as for Hadoop cluster installation, can be found in the following github repository:

```
https://github.com/project-asap/fabric-scripts
```

The main defined fabric tasks are the following:

- `bootstrap_wmt`: bootstrap WMT
- `bootstrap_IReS`: bootstrap IReS
- `bootstrap_spark`: bootstrap Spark Nested
- `bootstrap_operators`: bootstrap Swan, Telecom and Web analytics operators
- `bootstrap`: wrapper task for bootstrapping all the above components

The following sections describe the setup details and usage guidelines followed for creating the above tasks for each separate ASAP component.

4.3 WMT Setup

The WMT component uses Nginx [13], PHP-FPM [16] and Python, therefore these packages are among the components prerequisites. The following command installs these packages in an Ubuntu machine:

```
sudo apt-get nginx php-fpm python
```

The web content of the WMT is compiled using Grunt. The installation of Grunt's command line interface (CLI) globally can be done with the following commands:

```
sudo apt-get npm
sudo npm install -g grunt-cli
```

For building the component as a Grunt project, one needs to:

1. go to the component's root directory.
2. install project dependencies with `npm install`.
3. run Grunt with `grunt`.

Then the Nginx has to be configured to serve this content. Therefore a new file pointing to the WMT installation has to be created under the `/etc/nginx/sites_enabled`.

Finally, one can enjoy the WMT functionality using a web browser by navigation to the host and the port that Nginx uses to serve the content.

The workflow analysis and optimization functionalities can be challenged by running the following example script:

```
python pub/py/main.py analyse
python pub/py/main.py optimise
```

The Fabric tasks that automate the above procedure are listed in Appendix A.

4.4 IReS Setup

The IReS assumes a Hadoop [1] or Yarn [3] installation. Moreover, requires Maven [5] v3 for building the IReS components.

Running the IRes-Platform requires the following steps:

- Clone IReS-Platform:

  ```
  git clone https://github.com/project-asap/IReS-Platform.git
  ```

- Build `panic`, `cloudera-kitten` and `asap-platform` IReS components by navigating to the respective folder and running:

  ```
  sudo mvn clean install -DskipTests
  ```

- Update configuration files and folders appropriately.

After successful installation the file:

```
asap-platform/asap-server/src/main/scripts/asap-server
```

should be updated to set the `IRES_HOME` parameter to point to the location of the IReS installation.

Also the followind Hadoop/Yarn configuration files:

```
etc/hadoop/core-site.xml
etc/hadoop/yarn-site.xml
```

should be copied to the `asap-platform/asap-server/target/conf` directory.

The `asap-client` contains two examples for testing the main functionality for Operators and Workflows respectively. These examples can run using Maven:

```
mvn exec:java -Dexec.mainClass="gr.ntua.cslab.asap.examples.TestOperators"
mvn exec:java -Dexec.mainClass="gr.ntua.cslab.asap.examples.TestWorkflows"
```

The Fabric task that automates the above procedure is listed in Appendix B.

4.5 Spark Nested Setup

Spark Nested source code can be downloaded using the following `git` commands:

```
git clone https://github.com/project-asap/Spark-Nested.git
git checkout nested-hierarchical
```

Hence, it can be built using the SBT [17] following the guidelines for building the original Spark using SBT. Moreover, since we require Spark to read from HDFS, we need to build Spark against the installed HDFS version in our environment, for example:

```
sbt -Pyarn -Phadoop-2.7.1 assembly
```

After successful build a newly created jar should exist in the directory `assembly/target/scala-2.10`. Now, the cluster can be configured and start following the guidelines for the original Spark.

Test and benchmarks for challenging the extended functionality (nested map, hierarchical representation and distributed scheduling) can be found in this repository:

```
https://github.com/p01K/spark-tests
```

For building these examples:

- Create a `lib` directory and copy there the above created `spark-assembly-*.jar` file
- Build using SBT: `sbt package`

The examples can be run as Spark Applications by submitting it to the running spark master using the `spark-submit` script. For simplifying the execution the `submit.sh` can be used. This script expects as the first parameter the name of the submitting class followed by the parameters that should be passed to each class. The first parameter that the class expects is the URI of the Spark master, for example:

```
./submit NestedFilter1 spark://localhost:7077
```

4.6 ASAP Operators Setup

4.6.1 Swan Operators Setup

The Swan operators reside in the following repository:

```
https://github.com/project_asap/asap_operators
```

To compile and run tests, in the top-level directory one can run the following command:

```
make test
```

For completing successfully the compiling step the following compiler should be used:

```
icc version 14.0.0 (gcc version 4.4.7 compatibility)
```

Moreover, the repository at:

```
https://github.com/project-asap/swan_tests
```

includes directions, guidelines, and configurations required to install LLVM, clang, and the Swan runtime system.

4.6.2 Telecom Analytics Setup

The telecom analytics operators reside in the following repository:

```
https://github.com/project-asap/telecom-analytics/tree/develop
```

Currently, two use cases proposed in the scope of WP9 are implemented: the Peak detection and the Sociometer.

The Peak detection consists of a three separate operators: Data Filter, Distribution Computation and Peak Detection. All are implemented as spark applications in Scala. For building them:

- Copy the `spark-assembly-*.jar` file generated in 4.5 in the lib directory
- Build using SBT: `sbt package`

The documentation of these operators is publicly available at:

```
https://github.com/project-asap/telecom-analytics/blob
  /current/docs/PeakDetection.md
```

The `examples.PeakDetectionEx` application illustrates the whole use case execution, and can be run using the `submit.sh` script which undertakes to submit the application to the spark installation:

```
./submit.sh examples.PeakDetectionEx spark://localhost:7077
```

The Sociometer consists of three separate operators: User Profiling, Clustering and User Annotation. All are implemented as spark application in Python. The documentation of these operators is located here. They can run as spark applications using `pyspark`.

4.6.3 Web Analytics Setup

The web analytics operators developed for the purposes of WP8 reside in the following repository:

```
https://github.com/project-asap/web-analytics
```

They are implemented in Python and make excessive use of popular Python libraries for machine learning and scientific computing, such as: NumPy [14], pandas [15], sklearn, nltk, gensim The `command.sh` script demonstrates a flow of execution involving several of these operators.

4.7 Visualization API Setup

The visualization API source code resides in the following repository:

```
https://github.com/project-asap/ewrt
```

Testing code and examples of use are in the repository at:

```
https://github.com/project-asap/statistical-tests
```

The API is written in Python and can be installed along with their dependencies using the Python Setuptools [18]. The repository includes installation directions, a user guide, and example scripts.

4.8 Clusters

The ASAP prototype has been installed in the IMR and WIND clusters for the ASAP needs.

4.8.1 IMR Cluster

IMR cluster consists of 4 server-grade physical nodes. Each one of those is equipped with a 3rd generation i5 CPU (@ 2.90 GHz) and 16GB of physical memory and an array of two HDDs on RAID-0. The operating system is Debian 6 (squeeze) Linux. The cluster is equipped with Hadoop 2.6.0-cdh5.4.5.

4.8.2 WIND Cluster

WIND cluster consists of 4 server-grade nodes. Each one of those is equipped with a 24 Virtual CPUs and 24GB of memory. The operating system is Ubuntu 14.04 Linux. The cluster is equipped with Hadoop 2.7.1.

Chapter 5

Testing

We designed and use some tests to drive and evaluate the integration of ASAP modules into a working prototype. These often consist of subsets of the use cases described in D1.3 that involve more than one ASAP modules, along with a description of expected output and the requirements of each test.

5.1 Module Correctness Tests

Each partner has a number of unit tests to test their module.

5.1.1 IReS unit tests

The source code of IReS platform unit tests can be found at:

`https://github.com/project-asap/IReS-Platform/blob/master/asap-platform/asap-client/src`

The six unit tests for IReS platform are:

1. testCreateOperator: Create an operator object and test if its parameters are parsed and retrieved correctly.
2. testPutOperator: Create an operator object and insert it into the asap library using the rest client.
3. testRemoveOperator: Remove an operator from the asap library using the rest client.
4. testPutAndMatchOperator: Insert an abstract operator and check for its matching materialized operators.
5. testAddAbstractWorkflow: Insert an abstract workflow into the asap library using the rest client.
6. testMaterializeAbstractWorkflow: Materialize an abstract workflow according to a user defined policy.

5.1.2 Swan operators

The source code of the Swan operators unit tests can be found at: https://github.com/hvdieren/asap_operators/tree/master/tests
The seven unit tests are:

1. tfidf unit test: This unit test reads a simple user workflow description (tfidf.json) describing the operators, input and output datasets for calculating tfidf (term frequency inverse document frequency) values for each word in a corpus of documents. The workflow compiler will generate Swan code based on available operators as defined in the operators library (SwanMaterialised.json). The code will be compiled and executed to produce ARFF output in file "tfidf_output.arff". This output is compared against a "good" version and any deviances are reported.

2. kmeans unit test: This unit test reads a simple user workflow description (kmeans.json) describing the operators, input and output datasets for calculating k-means clustering from an ARFF text files containing TF-IDF values for words in a corpus of documents. The workflow compiler will generate Swan code based on available operators as defined in the operators library (SwanMaterialised.json). The code will be compiled and executed to produce a text output file in "kmeans_output.txt". This output is compared against a "good" version and any deviances are reported.

3. tfidf and kmeans unit test: This unit test reads a workflow description which contains a combined in-memory description of TF-IDF and K-means together. No output file is specified to TF-IDF and no input file is specified to K-means as the intermediate data is reatained in-memory. The code will be compiled and executed to produce a text output file in "tfidf_and_kmeans_output.txt". This output is compared against a "good" version and any deviances are reported. Timings gained from this benchmark are indicative of benefits of potential in-memory workflow optimizations.

4. tfidf then kmeans unit test: This unit tests reads a workflow description which contains a 2-phased transformation of a text dataset using TF-IDF "followed" by K-means. In contrast to "tfidf and kmeans", TF-IDF produces an output ARFF file which is specified as the input to K-means, when they produces the clustering results in text file "tfidf_then_kmeans_output.txt". As it is not benefiting from in-memory optimisations it is expected to take a longer time to execute.

5. tfidf standalone benchmark: This unit test compiles a version of TF-IDF which uses a list and an unordered_map datastructure, for benchmark comparisons against other versions of TF-IDF which use for example ordered map data structures. On execution it produces output in ARFF format (test_tfidf_list_umap.txt) which is compared against a "good" version and any deviances are reported.

6. kmeans standalone benchmark: This unit test compiles a standalone version of K-means using Swan for benchmark comparisons. It produces output in text format (test_kmeans.txt) which is compared against a "good" version and any deviances are reported.

7. wc standalone benchmark: This unit test compiles a standalone version of Word Count using Swan for benchmark comparisons. It produces output in text file listing of resulting word counts for a document (test_wc.txt) which is compared against a "good" version and any deviances are reported.

5.1.3 Dashboard unit tests

The source code of Dashboard unit tests can be found at: https://github.com/weblyzard/statistical-tests
It contains the following tests:

1. Three examples for POST validation: The examples (valid_observation.json, valid_observation2.json, valid_observation3.json) are available due to the fact that one can upload JSON files that contain observations with different fields. The webLyzard API defines a set of required and optional fields. The first example (valid_observation.json) only contains required fields: id, uri, added_date (indexing date), date (observation/document date), indicator_id, indicator_name, value). Some examples of optional fields are presented in the next example (valid_observation2.json): target_country, target_type, target_location, etc.

2. Two examples for invalid POST: The examples (invalid_observation.json, invalid_observation2.json) are available in order to highlight various types of errors. In the first example a fictional field (field that does not exists in the required or optional sets of fields described in the webLyzard API) named test error is defined which will make this test fail. The second example is expected to fail, as the id of the observation is missing.

3. PUT test: This example (update_observation.json) corresponds to an UPDATE statement. The expected output is a value of 2000 instead of 1000 for the first observation (id=1).

4. GET test: This test simply returns the data for a single observation.

5. DELETE test: This test simply deletes the data for a single observation.

5.1.4 Workflow Management Tool unit tests

The seven unit tests for the Workflow Management Tool are:

1. analyse: Compares the result with the presaved result in a file testwl-a.json.

2. save: Checks if save function generates a file with correct name.

3. execute: Checks if execute function saves a workflow in IReS format (correct folder and presence of required files in it).

4. findNode: Checks if the found node with findNode function has correct id.

5. findTask: Checks if the found task with findTask function has correct id.

6. findEdge: Checks if the found edge with findEdge function has correct id.

7. dict2text: Compares its result with the presaved result.

5.2 Integration Tests

Tests cases that challenge the interaction among the different ASAP components involve:

1. Integration of WMT and IReS: The WMT loads (using the IReS API) the available operators and they appear under the Tasks in the down left side of the WMT web application. A dummy workflow opened by the WMT can be executed over the IReS.

2. Integration of IReS and Swan: a dummy query that must be successfully started from within IReS and return the expected results.

3. Integration of IReS and Spark : A dummy query that must be successfully started from within IReS and return the expected results.

4. Integration of Telecom Analytics application with Visualization Tool: a dummy script that uploads data to the visualization tool (using the weblyzard API) and they appear in the Visualization Dashboard.

5.3 Jenkins

Test scripts for the test cases described above can currently run on demand. Ongoing work aims to extend them in order to cover additional functionality as the components evolve and also to be configured to run periodically and in automated way over Jenkins [9]. However, since many of the components have a web interface, currently, some integration tests can only be performed manually. We are currently investigating more sophisticated, automated test practices for the web-based tests.

We have installed Jenkins 2.6 for continuous integration of the separate ASAP components. It resides in a machine connected to the FORTH's private network. We have created jobs for building and testing each component separately. Jobs are configured to run weekly and the build status is sent via email to the developers of each component along with the build logs. For the components with elementary setup we created "freestyle" Jenkins jobs, while for those having more complex workflows (IReS, Spark and Swan) we used pipelines built with simple text scripts that use a Pipeline DSL (domain-specific language) based on the Groovy programming language. For meeting the particular requirements of the above jobs we have extended Jenkins by installing among others the following plugins:

1. ShiningPanda Plugin — https://wiki.jenkins-ci.org/display/JENKINS/ShiningPanda+Plugin — Jenkins support with Python and virtualenv builder

2. NodeJS Plugin — https://wiki.jenkins-ci.org/display/JENKINS/NodeJS+Plugin — Jenkins integration for NodeJS and npm packages

3. Make Plugin — https://wiki.jenkins-ci.org/display/JENKINS/CMake+Plugin — Jenkins support for make projects

4. Maven Project Plugin — https://wiki.jenkins-ci.org/display/JENKINS/CMake+Plugin — Jenkins support for Maven jobs

5. sbt plugin — https://wiki.jenkins-ci.org/display/JENKINS/sbt+plugin — Jenkins support for sbt projects
6. Git Plugin — https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin — use of Git as a build SCM

Bibliography

[1] Apache hadoop. https://hadoop.apache.org.

[2] Apache hadoop hdfs. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[3] Apache hadoop yarn. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[4] Apache kitten. https://github.com/cloudera/kitten.

[5] Apache maven. http://maven.apache.org/.

[6] Elasticsearch. http://www.elasticsearch.org/.

[7] Fabric. http://www.fabfile.org.

[8] Javascript. http://javascript.com.

[9] Jenkins. http://jenkins-ci.org/.

[10] Jersey. https://jersey.java.net/.

[11] Jetty. http://eclipse.org/jetty/.

[12] Mahout. http://mahout.apache.org.

[13] Nginx. http://nginx.org.

[14] Numpy. http://www.numpy.org.

[15] pandas. http://pandas.pydata.org.

[16] Php-fpm. http://php-fpm.org.

[17] Sbt. http://www.scala-sbt.org.

[18] Setuptools. http://pythonhosted.org/setuptools/.

[19] Weka. http://weka.wikispaces.com.

[20] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: an efficient multithreaded runtime system. In *PPoPP*, 1995.

[21] H. Vandierendonck, G. Tzenakis, and D. S. Nikolopoulos. A unified scheduler for recursive and task-based parallelism. In *PACT*, 2011.

Appendix A

WMT Setup Fabric example

```
@task
def install_npm():
    try:
        run("npm version")
    except:
        sudo("apt-get install npm")
    user = os.environ['USER']
    group = run("groups|cut -d ' ' -f 1")
    with quiet():
        sudo("chown -fR %s:%s ~/.npm ~/tmp" % (user, group))

@task
def install_grunt():
    # install grunt-cli
    sudo("npm install -g grunt-cli")
    if not exists("/usr/bin/node"):
        # create symbolic link for nodejs
        sudo("ln -s /usr/bin/nodejs /usr/bin/node")

@task
def config_nginx():
    sites_available = "/etc/nginx/sites-available/%s" % VHOST
    sites_enabled = "/etc/nginx/sites-enabled/%s" % VHOST
    sudo("echo \"%s\" > %s" % (VHOST_CONFIG, sites_available))
    if not exists(sites_enabled):
        sudo("ln -s %s %s" % (sites_available, sites_enabled))

@task
def install_nginx():
    sudo("apt-get install nginx")
    config_nginx()

@task
def start_nginx():
    sudo("nginx -s reload")

@task
def test_wmt():
    content = run("curl http://localhost:%s" % WMT_PORT)
    assert("workflow" in content)

@task
def bootstrap_wmt():
    # install prerequisites
    install_npm()
    install_grunt()

    # clone and build wmt
    install_wmt()
    install_nginx()
    start_nginx()

    test_wmt()
```

Appendix B

IReS Setup Fabric Task

```
@task
def bootstrap_IReS():
    def build():
        # Conditional build
        if not exists("asap-platform/asap-server/target"):
            for d in ("panic", "cloudera-kitten", "asap-platform"):
                with cd(d):
                    run("mvn clean install -DskipTests")

    install_mvn()

    clone_IReS()

    with cd(IRES_HOME):
        build()
        # Update hadoop version
        HADOOP_PREFIX, HADOOP_VERSION = check_for_yarn()
        for f in ('asap-platform/pom.xml', 'cloudera-kitten/pom.xml'):
            change_xml_property("hadoop.version", HADOOP_VERSION, f)
        # Set IRES_HOME in asap-server script
        run_script = "asap-platform/asap-server/src/main/scripts/asap-server"
        c = run("grep \"^IRES_HOME=\" %s | wc -l" % run_script)
        if (c == "0"): # only if it is not already set
            run("sed -i '/#$IRES_HOME=$/a\IRES_HOME=%s' %s" % (IRES_HOME,
                                                               run_script))
        for f in ("core-site.xml", "yarn-site.xml"):
            sudo("cp %s/etc/hadoop/%s "
                 "asap-platform/asap-server/target/conf/" % (HADOOP_PREFIX, f))
    start_IReS()
    test_IReS()
```

FP7 Project ASAP
Adaptable Scalable Analytics Platform

End of ASAP D7.2 Integration Prototype "ASAP System Prototype v.2"

WP 7 – Integation of the ASAP System

Nature: Report

Dissemination: Public