



# Demonstrator and report on workflow compilation and parallel execution


## Authors

Alan Akbik (Technische Universität Berlin)

Peter May (British Library)

November 2013

*This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).*

*This work is licensed under a CC-BY-SA International License* 



## Executive Summary

Preservation of digital media collections has become a scalability issue due to the large quantity of heterogeneous content. The amount of information has reached a point where distributed, shared nothing computing environments are necessary to enable processing in a timely manner. Hadoop is a system that abstracts from the hardware in a computing cluster and presents the user with a programming interface. The user can load data into the system and write programs that execute on the cluster, while not worrying about network transfers or data access. At the same time, governmental organizations like public libraries want to have an intuitive yet powerful graphical user interface to design preservation workflows that are supposed to run on Hadoop, or other scalable processing environments.

In this deliverable, we investigate two possible solutions for the scalable execution of preservation workflows. Firstly, we report on the automatic compilation of Taverna workflows to MapReduce jobs for parallel execution, and demonstrate the compilation of a simple example workflow. Secondly, we report on the use of a higher order dataflow language, namely Apache PIG, to define preservation workflows. We discuss a complex workflow that is being developed for preservation watch on large scale data. We discuss advantages and disadvantages of both approaches, especially with regards to the issues of workflow elements that can effectively be parallelized and optimized. Based on this analysis, we give an outline of a best-of-both worlds approach to bring together the graphical interface of Taverna with the optimized workflow execution and maintainability of Apache PIG.

## Table of Contents

Executive Summary		iii
1	Introduction	1
1.1	Goals of the Execution Platform	1
1.2	Deliverable D6.1	1
1.3	Contributions of this Deliverable	2
2	Defining and Compiling a Workflow in Taverna	3
2.1	The PPL-Translator	3
2.2	Translation Process	3
2.3	Demonstration Workflow	4
2.4	Identified Issues	5
2.4.1	User Defined Functions	5
2.4.2	Workflow Operators	6
2.4.3	Workflow Optimization	6
2.5	Conclusions	7
3	Complex Workflow in Apache PIG	7
3.1	Preservation Watch Workflow	7
3.1.1	Use Case Motivation	8
3.1.2	Workflow Overview	8
3.2	Choice of Apache PIG	9
3.3	Implementation Example	10
3.4	Workflow Execution	11
3.5	Advantages of Defining Workflows in Apache PIG	11
4	Proposed Solution	12
4.1	Summary of Investigated Approaches	12
4.2	Integrated Approach	13
4.3	Open Questions	13
5	Conclusion	14
6	References	14

# 1 Introduction

In this chapter, we first give an overview of the goals of the execution platform in the SCAPE project. We then give a short overview of the status of the work in WP6 and list the contributions of this deliverable.

## 1.1 Goals of the Execution Platform

The goal of the execution platform is to provide a general means of facilitating the specification and evaluation of complex preservation operations over very large volumes of data. Because preservation in the SCAPE context is generally performed by public organizations such as national libraries, a principal goal of WP6 is to make the process of developing complex preservation workflows that scale to distributed processing environments as easy as possible.

To this effect, we investigate different tools and higher order languages that enable the formulation of complex workflows with regards to the following main *desiderata*:

**Ease-of-use.** Firstly, the formulation of complex and scalable workflows should not require an extensive background in distributed processing technologies. Rather, it should be feasible for average persons with a background in computer science to create, modify and execute scalable preservation workflows. This requires abstraction from the commonly used MapReduce paradigm as this is not intuitive to persons without a background in distributed processing technologies.

**Efficient execution.** Secondly, even complex workflows should be executed in an optimized way to insure that the available hardware resources are efficiently used. This desideratum derives from use cases in digital preservation that require heavy computation. We see the need for optimization as of major importance for the efficient execution of preservation workflows.

**Maintainability.** Thirdly, the underlying technologies should be well-established and well-maintained. A solution that rests on projects that are maintained, and constantly improved, by an active community will allow it to be maintainable after the conclusion of the SCAPE project.

The solutions we present in this deliverable will be examined with regards to these desiderata.

## 1.2 Deliverable D6.1

In Deliverable D6.1, we presented the PPL-translator that compiles Taverna workflows to MapReduce, and conducted a set of experiments to investigate the feasibility of executing preservation workflows on a Hadoop [Bor07] cluster. We compared setups in which a workflow was defined in Taverna [OAF+04] and compiled down to either a series of MapReduce [DG08] jobs or executed as a whole in one Map job. In addition, we compared these setups with a manually implemented MapReduce job. In our experiments, we found manual creation of MapReduce jobs to be significantly faster in execution than the automatic compilation of Taverna workflows into MapReduce. One factor responsible for the difference in runtime is that the automatically compiled

version employs beanshells to execute the logic, meaning that every Reduce call creates a new beanshell to parse the script.

We also noted possible optimization methods in the deliverable: These include either creating a single beanshell per Reduce class (not per call of Reduce) or to move the logic from the Reduce phase into the Map phase to enable the execution of a higher number of concurrent tasks. However, the downside of the latter approach is that creating dot and cross products would require an additional MapReduce phase before executing the actual logic of the activity. Generally, processing overhead must be minimized by jointly executing all processes that can be bundled into one execution of a Map or Reduce phase.

### 1.3 Contributions of this Deliverable

Examining the results from the work reported on in Deliverable D6.1, we noted a number of problems for automatic workflow compilation to MapReduce given the *desiderata* defined in Section 1.2. *Firstly*, most Taverna workflows currently in use by the Taverna and SCAPE communities define user defined functions (UDFs) in ways that run contrary to the MapReduce paradigm. *Secondly*, only a subset of all operators in Taverna can be compiled to the MapReduce paradigm. And *thirdly*, the compilation and optimization of complex workflows is highly difficult and costly in terms of development effort. We address all three points in detail in this deliverable.

We focus especially on the point of compilation and optimization. In related projects, optimization methods are subject to heavy research and development efforts. Prominent examples are Apache PIG [ORS+08], Spark [ZCF+10], Hive [TSJ+09] and many others. We believe that a comprehensive solution to the problem of defining and optimizing scalable preservation workflows can be achieved by building on research and development efforts made by such projects.

In this deliverable, we investigate two possible solutions for the scalable execution of preservation workflows, one based on the existing work with the PPL translator, and the other with defining complex workflows in Apache PIG. This deliverable therefore consists of three main contributions.

**Taverna workflow compilation and demo.** We describe the current state of the PPL-translator which automatically compiles Taverna workflows into MapReduce jobs, and discuss a demonstration workflow. We specifically discuss issues with regards to compiling Taverna to MapReduce, and discuss advantages and disadvantages of the Taverna and PPL-translator approach.

**Execution of a complex preservation workflow in Apache PIG.** Secondly, we introduce a much more complex workflow from the area of preservation watch, and show how it was implemented in the Apache PIG workflow language. We use this workflow to discuss similarities of workflow elements both in Taverna and Apache PIG, and observe advantages of the latter.

**Discussion of both approaches and outline for future work.** Thirdly, we discuss advantages and disadvantages of both approaches and make the case for compiling Taverna workflows not directly to MapReduce jobs, but to Apache PIG workflows, integrated within Apache Oozie scheduling. We believe that this will greatly improve maintainability and flexibility of the project, while allowing Taverna workflows to gain access to the workflow optimization techniques of Apache PIG.

The structure of this deliverable follows the three above mentioned points. In Chapter 2, we describe the PPL-translator and give an overview of an example demonstration workflow. In Chapter 3, we introduce a complex preservation watch workflow and discuss its implementation using Apache PIG. Finally, in Chapter 4, we discuss strengths and weaknesses of both and give an outline of a best-of-both worlds approach to bring together the graphical interface of Taverna with the optimized workflow execution and maintainability of Apache PIG.

## 2 Defining and Compiling a Workflow in Taverna

In this Chapter, we investigate the process of defining a preservation workflow in Taverna and automatically compiling this workflow to MapReduce using the PPL-translator. We discuss a number of limitations of this approach in detail. These issues are the basis of considerations in Chapter 3 and are addressed in our proposed solution outlined in Chapter 4.

### 2.1 The PPL-Translator

In Deliverable D6.1, we presented an overview of the "Program for parallel Preservation Load" (PPL). Its purpose is to enable the parallel execution of preservation workflows. Essentially, the program takes a workflow as input and automatically generates a class that can be uploaded to and executed by Hadoop.

The resulting Hadoop execution is a linear list of MapReduce jobs produced from the input workflow. The required input for each individual job is either read locally (as provided by the Hadoop framework), or is fetched from other machines in the Hadoop distributed file system (HDFS), if required. The translator can either be used as a compiled jar file, or can be built from source.

### 2.2 Translation Process

Figure 1 shows a simplified version of the translation process. PPL uses the SCUFL2 API to interpret workflow files created with Taverna. The translator starts at the workflow's output ports and follows the data links that connect the workflow elements backwards and recursively, with the goal of building a linear list of workflow elements. This process is repeated until the workflow's input ports are reached and all workflow elements have been added to the list. This list is then reversed to reflect the actual sequence of workflow elements in the order of dependence, meaning that if an element A depends upon element B, element B appears first in the list.

Next, each element in the list is translated individually from a template. For activities with multiple inputs, the Taverna user decides whether to combine these using a cross product or a dot product; the resulting Hadoop job also includes this logic. Since the execution of the activity must be done after the dot or cross product is created, activity logic is implemented in the Reduce phase. In this manner Map and Reduce can be used to build the dot or cross product before invoking the activities (a dot product for example can be implemented as a reduce side join).

An activity in the Taverna workflow that is to be translated can have any number of input ports and any number of output ports. The data location of outputs from output ports is chosen based on the

input ports it feeds in to. In turn, the reader that reads the data for those input ports looks in those locations and reads the data. The location consists of a prefix defined by the user, the activity's name, and the input port's name.

Finally, Java creates a JAR that can be executed on Hadoop. This JAR includes all dependencies, so that they are available to Hadoop.

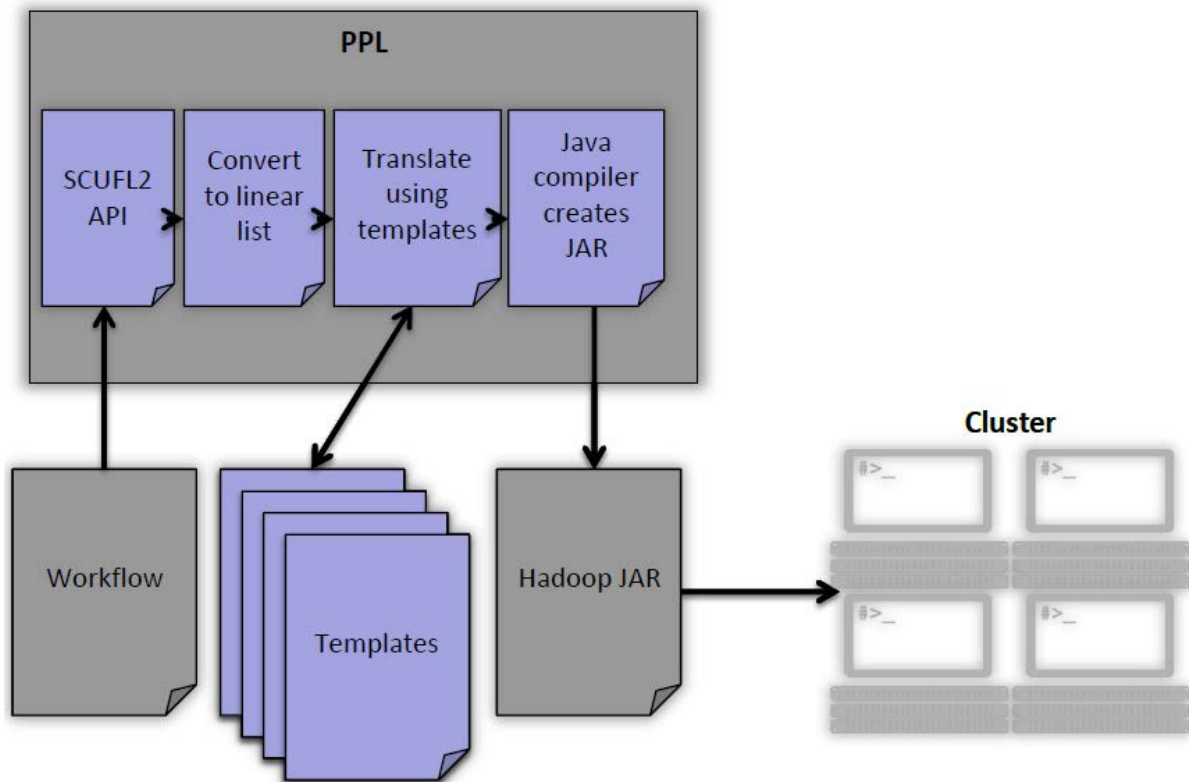


Figure 1: A schematic overview of the workflow compilation process in PPL.

### 2.3 Demonstration Workflow

We defined a simple demonstration Taverna workflow that can be compiled to MapReduce using the PPL, consisting of two beanshells. The first processes the workflow input removing all characters after the first space. The second appends the letters “bench” to each line. We tested the job using a 6GB file containing almost 80 million lines of random text. We report on the results in Deliverable D6.1.

The demo and instructions for its execution have been made available online in our GitHub repository at <https://github.com/schenck/taverna-to-hadoop>. Developers can download the source code of the PPL translator and follow the instructions to execute the example workflow locally.

## 2.4 Identified Issues

In this section, we list principal issues that have been identified with regards to using Taverna as a tool for defining scalable and executable workflows. In brief these include:

**User defined functions.** Taverna workflows make use of Web services and calls to locally installed tools in place of user defined functions, both approaches of which do not lend to parallelization by cloud computing. This is discussed in detail in Section 2.4.1.

**Workflow elements.** Taverna is much more expressive in terms of workflow elements than the dataflow elements that can be expressed with the MapReduce paradigm, therefore limiting the types of workflows that can be effectively parallelized. This is discussed in detail in Section 2.4.2.

**Workflow optimization.** As shown in Deliverable D6.1, the compilation of single fixed dataflows is not yet optimized, causing decreases in execution time. Without workflow optimization, we believe that the execution time of even moderately complex workflows will be heavily impacted. This is discussed in detail in Section 2.4.3.

We focus on these issues in detail in the following subsections. Our proposed solutions are discussed in detail in Chapter 4, after we first discuss a highly complex and scalable workflow we implemented in Apache PIG in Chapter 3.

### 2.4.1 User Defined Functions

User defined functions (UDFs) are components that perform custom processing on data and are usually embedded and connected to each other in a workflow by dataflow elements. In the SCAPE project, UDFs will most often execute SCAPE Components. The most common ways of using UDFs in Taverna is by calling one of the many Web services listed in Taverna's repository. The advantages of this are obvious: Users need not install set of third-party tools and software on their local machines to compose and execute a workflow, meaning that users need not face issues such as compatibility of tools with their operating systems and other libraries. This makes Taverna a comfortable tool for its main target community, namely users without an extensive background in computer science.

However, a drawback of this approach is that it does not lend to parallelization by cloud computing, at least not from the client side. Because the Web services that are called lie outside a local Hadoop cluster, parallel calls to these services will not result in any gains in execution speed. Instead, this introduces a number of risks into parallel execution, such as fault tolerance against temporarily unavailable or overloaded Web services. This issue has been recognized by the SCAPE community, which has agreed that scalable preservation workflows should not include calls to external Web services.

Another proposed method of defining user defined functions is local tool invocations, i.e. the invocation of software that is installed on a local machine. In this scenario, the Taverna workflow calls a local tool, inputs data, and retrieves the output of the tool to be passed to the next UDF. While this method will enable parallel execution, it requires an entire cluster of machines to be set up with exactly the same suite of tools for a given workflow. This moves drastically away from the above



mentioned scenario of the typical Taverna user having limited computer skills, as this no longer enables users to compose and execute workflows without extensive work in setting up each node of the cluster. In this scenario, a set of tools must be installed on an entire cluster of machines, either by the user or - more probably - by an experienced network administrator. Worse, this costly process might need to be repeated for each workflow with different tool requirements, or even changes in individual tools.

### **2.4.2 Workflow Operators**

As Taverna was developed without parallelization through MapReduce in mind, an important issue was identifying the scope of workflow operators that can effectively be expressed in the MapReduce paradigm. One of the main advantages of this paradigm, namely the ease of use with a simple abstraction, also limits its application to single fixed dataflows, i.e. programs that read a single input and generate a single output. This means that many more complex algorithms and workflows, particularly those with multiple inputs, are difficult to implement in a pure MapReduce environment [LLC+12].

Beyond its missing support for multiple inputs, MapReduce does not record global state information during processing. This affects workflow elements such as loops (i.e. cycles in workflow definitions) which requires such information for execution and termination. There also is no support for decisions (i.e. switch-case statement enabling a workflow to make a selection on the execution path to follow) or forks (which split one path of execution into multiple concurrent paths of execution). Finally, the popular operator Join is not well supported by Map and Reduce functions, since the paradigm is designed for processing a single input. However, several solutions exist, such as the Map-side join and the Reduce-side join that enable joins to be expressed in MapReduce [LLC+12].

These limitations of the MapReduce paradigm impact our considerations of what is possible in terms of automatic Taverna workflow compilation. Our existing prototype of the PPL-translator accordingly can handle only those workflows that are within these limitations, namely single fixed dataflows. As we noted above, other workflow elements such as joins and multiple inputs are not supported, yet an implementation in MapReduce is possible, however with considerable effort in research and development. Having recognized these limitations, the research community has been actively developing solutions that either build on MapReduce [ORS+08], complement it [IHB+12] or go beyond it [EST+13].

In light of the limitations we have outlined, as well as the desiderata we have outlined in Section 1.1, we believe that we can develop a solution by building on the wealth of work that is being done by the research community. In Chapter 3 we examine such a technology and note implications for our work.

### **2.4.3 Workflow Optimization**

As noted, the PPL translator generates a linear list of MapReduce jobs that sequentially execute each step of the defined workflow. This type of naive execution has severe disadvantages in runtime, as our experiments showed in Deliverable D6.1. The reason for this is an unnecessarily large number of MapReduce executions resulting in a significant processing overhead; ideally, all processes that can

be bundled into one execution of a Map or Reduce phase should be executed jointly. The potential for optimization becomes more significant as the workflow complexity grows - however, at the same time the complexity of determining good optimization procedures grows as well.

Much current research is devoted to this problem. In the Stratosphere project [LAS+13], for example, a workflow is defined using higher order data flow elements, and the entire workflow is optimized before compilation. However, users must also add specific annotations to workflow elements, which in turn requires users to have knowledge about the underlying execution platform. This somewhat contradicts the principal desideratum we defined in Section 1.1, namely ease-of-use, as we would like to make optimized workflows available to users without a background in parallel processing.

Implementing an optimizer specific to Taverna is something that can be attempted; however we believe the effort involved to be not justified given the expected outcome.

## 2.5 Conclusions

In this chapter, we have investigated the process of defining a preservation workflow in Taverna, using the PPL-translator to compile it to a MapReduce program and execute it on a cluster. We have noted a number of issues with regards to user defined functions, as well as workflow elements that can be defined in Taverna, but not compiled to MapReduce. We believe that a direct compilation of a subset of dataflow elements in Taverna into an optimized sequence of Map and Reduce steps is something that can be researched, however we believe the effort involved to be not justified given the expected outcome.

We also note that many of these issues have been recognized by a large and active research community in the field of scalable processing. Significant process has been made with mature open source technologies that an optimized scalable workflow execution may be built on. We therefore investigate technologies such as Apache Oozie and Apache PIG in the following chapter and proceed to propose a solution to workflow compilation that leverages these technologies.

## 3 Complex Workflow in Apache PIG

In this chapter, we investigate the use of a mature open-source technology, namely Apache PIG, for the parallelization of a highly complex workflow which relates to preservation watch. We introduce the workflow and give details on its implementation using the Apache PIG dataflow language. We then discuss implications for our work in automatic workflow compilation.

### 3.1 Preservation Watch Workflow

We introduce a complex, scalable workflow that is implemented within the SCAPE project. The objective of this workflow is to process very large collections of Web pages (stored in Warc-files) and extract specific types of information from unstructured data that are of relevance to the preservation watch component in order to determine preservation risks.

### 3.1.1 Use Case Motivation

In this use case, we address an information need from the National Library of the Netherlands, which aims to create an up-to-date repository of e-Journals and publishers in order to address questions of ownership and terms of access of such digital objects. The National Library of the Netherlands addresses the problem that the shift to journal content that is digital, online and held remotely has challenged the responsibility that libraries have in ensuring the continuity of access to these materials.

The motivation of this workflow therefore is to increase the ability to recognize when digital content is becoming endangered, while addressing the challenge that the processes of gathering, managing and reasoning on relevant knowledge can become manually infeasible when the volume and heterogeneity of content increases, multiplying the aspects to monitor.

To address this challenge, we proposed and implemented a system that automatically gathers such information from Web sources by performing Information Extraction on a very large scale. We recently presented the use case and the implemented preservation watch workflow at [FAS+13]. For a full discussion of the use case, refer to [FAS+13].

### 3.1.2 Workflow Overview

The system consists of a sequential workflow of a number of modules. We give a schematic overview in Figure 1. More detailed explanations on the system design can be found in [AVH+12], as well as [AVKL13] and [AL12]. The main modules are as follows:

**Data input.** A data input module which reads Warc-files (among other formats) into the workflow. In our experiments, we use the ClueWeb09 corpus [CCS09], a 25 terabyte Web crawl that contains over 1 trillion Web pages. The ClueWeb09 was chosen because of its size and because it is the main reference corpus for scalable analytics of Web pages.

**Data pre-processing.** We run a pipeline of pre-processing steps to analyse the data. This pipeline executes a series of steps including *spam detection* [CSC11], in order to remove spam Web pages from the corpus, *boilerplating*, to remove unnecessary HTML code, entity detection, to find all entities of interest, as well as a complete *natural language processing* pipeline for syntactic analysis. This process is extremely computation intensive and requires parallel execution. We require these data pre-processing steps to enable data and feature extraction.

**Data and feature extraction.** We perform a feature extraction step to find all data points that are potentially relevant to the information need given by the use case and characterize these data points using a set of features derived from the syntactically annotated data. This process is explained in detail with examples in [AVH+12] and [AVKL13].

**Data mining.** On this representation, we execute data mining algorithms, which consist of two steps. The first is the computation of a distance matrix (also referred to as dissimilarity matrix) that indicates the distance between each data point as measured by the Cosine distance of their respective feature vectors. This matrix then enables us to execute unsupervised data mining

algorithms, such as hierarchical agglomerative clustering (HAC) on this data to discover classes of related data points.

The results of the data mining pipeline are then passed on to the *Propminer* [AKM13] tool, which enables us to perform Information Extraction on the data set. The very large amounts of data involved require this workflow to be executed in a distributed processing environment. Furthermore, the complexity of this workflow requires the process of formulating and implementing it to scale, so that modifications can be easily made and different workflows be evaluated.

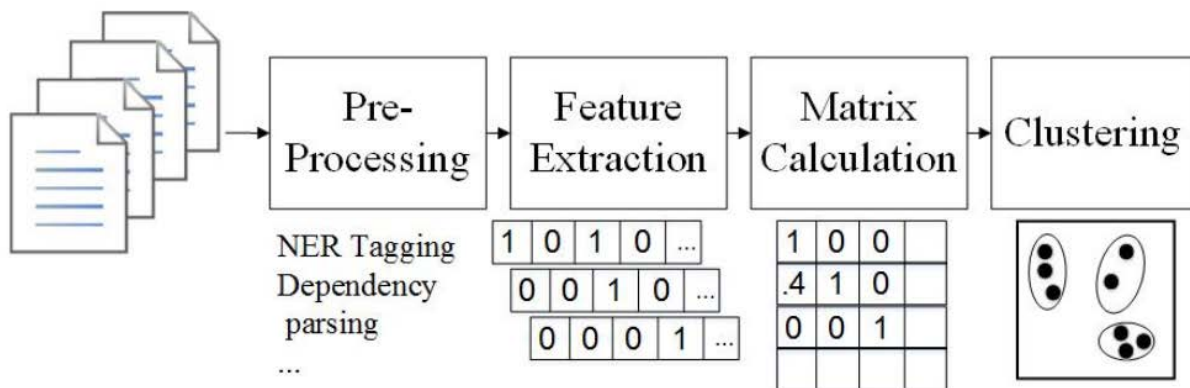


Figure 2: A schematic overview of the data mining workflow implemented in PIG.

### 3.2 Choice of Apache PIG

Early attempts of the implementation were focused on directly implementing the workflow a series of MapReduce jobs, but after much development work it became apparent that this was not feasible due to its size and complexity. Using Taverna to define the workflow was also not feasible, as the automatic compilation of Taverna workflows was in a prototypical state at the time of development, and - as noted in chapter 1 - the automatic optimization of distributed workflows is a complex and effort intensive process.

Instead, we chose to implement the workflow in the Apache PIG language, a higher order data flow language that supports a nested data model. This simple-to-use scripting language enables developers to create scripts that automatically compile down to a series of MapReduce jobs. This is done by generating a logical plan of the workflow execution, performing optimization techniques that are adopted to the compilation. Since Apache PIG is built on top of Hadoop, its usage requires no modification to Hadoop, and workflows can be immediately executed in a distributed environment.

This provided us with a number of advantages; Firstly, PIG provides a set of data aggregation operators, such as GROUP\_BY, JOIN and COUNT and has the ability to execute arbitrary user defined functions (UDF) in programming languages such as Java, Python and JavaScript. Much of the complexity we noted in Section 2.4.2 of compiling for example JOINS to MapReduce is therefore already handled in PIG. Secondly, PIG scripts support dataflows with multiple inputs, allowing us to formulate more complex dataflows than is possible in MapReduce without significant effort. Thirdly, PIG is widely supported; for example, it is bundled with the popular *Cloudera* distribution of Hadoop

and cluster management tools, and is directly supported by the workflow execution system Apache Oozie.

```
-- Script to parse WARC records and join with waterloo spam records
DEFINE Clueweb09RecordLoader edu.tuberlin.corpora.clueweb.input.pig.Clueweb09RecordLoader();
DEFINE SentenceSplitter edu.tuberlin.dima.nlp.pig.SentenceSplitterFunction('en');

DEFINE BoilerPipe edu.tuberlin.dima.nlp.pig.BoilerPipeFunction();
DEFINE LanguageDetector edu.tuberlin.dima.nlp.pig.LanguageDetectorFunction();

-- input clueweb09 corpus
Clueweb09 = LOAD 'src/test/resources/edu/tuberlin/corpora/clueweb/input/ClueWeb09_English_Sample.warc'
            USING Clueweb09RecordLoader() AS (uri:chararray, warc_html:chararray, trecid:chararray);

-- input waterloo spam corpus
Waterloo_spam = LOAD 'src/test/resources/edu/tuberlin/corpora/clueweb/input/waterlooSample'
                USING PigStorage(' ') AS (percentile_score:int, clueweb_docid:chararray);

-- filter spam articles
Waterloo_spam_filtered = FILTER Waterloo_spam BY percentile_score > 70;

-- joining both corpora on trecid
Clueweb09_without_spam = JOIN Clueweb09 by trecid, Waterloo_spam_filtered by clueweb_docid;

-- now boil the text
boiled = FOREACH Clueweb09_without_spam GENERATE trecid, uri, BoilerPipe(warc_html) as boiled;

-- match the language
filtered = FILTER boiled by LanguageDetector(boiled) == 'en';

-- extract sentences text
elements = FOREACH filtered GENERATE trecid, uri, SentenceSplitter(boiled) as s;
sentences = FOREACH elements GENERATE trecid, uri, flatten(s);
```

**Figure 3: Example PIG script.**

### 3.3 Implementation Example

We give an excerpt of the Apache PIG script for the data input and data processing steps in Figure 2. The goal of the script is to read two inputs, the ClueWeb corpus and the Waterloo Spam Ranking corpus. Both inputs are then joined, so that each Web page in the ClueWeb corpus has a spam value, which indicate the probability that this Web page is spam. The script filters out all Web pages that are not in English and have a high probability to be spam. On the remaining Web pages, some pre-processing steps are performed.

We discuss this script in more detail, line by line:

**Define.** The first four lines register UDFs that we call in the script. The UDFs were written in Java and each take an input and produce an output. Some UDFs, such as SentenceSplitterFunction make take constructor arguments, such as the language it should process.



**Load.** The following two lines are loading functions. The first executes a loading function that reads the Warc files from disc using a Warc-file reader from the *Cloud9* project. The second line reads the Waterloo Spam Ranking repository from the file system.

**Filter.** We then filter the spam ranking repository to remove all entries that have a high probability of being spam.

**Join.** We then join the filtered spam ranking repository with the ClueWeb corpus, thus removing all Web pages from the ClueWeb corpus that have a high probability of being spam.

**Process.** On the filtered ClueWeb corpus (referred to as *clueweb\_without\_spam*), we perform boilerplating to remove all HTML. We then detect the language of each Web page and filter out all pages that are not in English. Finally, we use the sentence splitter to extract full sentences from all remaining Web pages.

### 3.4 Workflow Execution

We executed the workflow on a 9 node Hadoop cluster, each node with 24 Cores, 256 GB RAM, 12x2TB disks running CDH 4.1.0 on top of Ubuntu 12.04. The syntactic analysis of the text in the pre-processing step was the most costly, requiring over 5600 CPU hours. By contrast, feature extraction was performed in 3 CPU hours. The execution time of the ensuing data mining tasks depends on the filters and clustering algorithms involved.

All in all, we note that very complex workflows consisting of dozens of Map and Reduce steps can be defined with a powerful abstraction layer and executed in a distributed environment.

### 3.5 Advantages of Defining Workflows in Apache PIG

There are four main advantages of using Apache PIG, namely the intuitive abstraction layer, integrated workflow optimization techniques, an independence of the underlying execution platform and a large and active community of developers that maintain and expand the project. We give details on each of the points in the following:

**Abstraction layer.** The abstraction layer is a simple-to-use script-like way of defining data flows. Developers need not think in terms of Maps and Reduces but rather simply a series of joins, selects and calls of UDFs (user defined functions). The code example above highlights the readability of the code. The developed system is of very high complexity, compiling down to dozens of Map and Reduce jobs, yet use of the abstraction layer enabled rapid and flexible development of the system.

**Independence of execution platform.** The underlying execution platform is by default Hadoop and MapReduce, but compilers are becoming available for a growing number of other paradigms. One example is a compiler for Spark (available at <https://github.com/mateiz/spork>) being developed at Twitter. Another example is a compiler that allows PIG scripts to be executed on the Stratosphere platform [KVB13]. Any program written in PIG is therefore independent of a specific execution platform, making it more robust against possible paradigmatic shifts in distributed processing.

**Workflow optimization.** Apache PIG workflows are optimized before being translated to MapReduce, a process that is continuously being improved on by a large community of developers. Workflow optimization is not something the user of the abstraction layer needs to be concerned with, making the process of defining optimized workflows easier for persons without an extensive background in distributed processing.

**Community of developers.** There is a large and active community of users that maintain and extend the expressive power of Apache PIG, workflow optimization, and support for hardware architectures. Apache PIG has become the main higher order dataflow language for defining complex scalable workflows. By using this higher order language, preservation workflows defined in the SCAPE project can leverage the substantial research and development efforts made by this community.

Nevertheless, the process of writing user defined functions for Apache PIG requires persons to have a programming background in at least Java or Python. Workflows are defined in a script language that data scientists may find less intuitive than a graphical user interface. While Taverna provides a graphical interface that users may find more intuitive, it is not suited for data scientists with no programming abilities, as some control elements need to be implemented in Java beanshells. Taverna users have noted that the process of creating large and complex workflows with Taverna requires much effort. Feedback from the SCAPE community however indicates that most preservation workflows will not be overly complex in terms of dataflow elements, indicating that Taverna workflows may be a suitable representation for preservation workflows. We explore this further in the next section.

## 4 Proposed Solution

In this chapter, we summarize our observations on the two solutions for defining scalable preservation workflows and propose an approach that addresses many of the issues that we have identified and noted in this deliverable. We give an overview over the proposed solution, discuss open questions and outline present and future work.

### 4.1 Summary of Investigated Approaches

We presented two solutions for defining preservation workflows that can be compiled to scalable processing environments. On the one hand, we presented a compiler prototype that given a Taverna workflow generates a sequence of MapReduce jobs to be executed on the Hadoop platform. On the other hand, we demonstrated a very complex workflow defined in the Apache PIG script language, which compiles down to an optimized sequence of MapReduce jobs. We noted advantages and disadvantages of both approaches.

**Taverna.** Strong points for the first approach using Taverna are the graphical user interface and an active community of users of this tool. However, we note that Taverna is not suited for data scientists with no programming abilities, as some control elements need to be implemented in Java beanshells. Especially the process of creating large and complex workflows with Taverna requires much effort. However, the main disadvantages of compiling down Taverna workflows directly to



MapReduce jobs is the infeasibility of compiling and optimizing workflows that contain multiple inputs or workflow elements such as joins, loops and forks, as well as defining user defined functions in a way that enables parallel execution. These limitations push a number of large scale preservation workflows out of reach for most hardware environments. In addition, because the PPL-translator directly compiles to MapReduce, our solution is tied to only one of several emerging options of distributed processing.

**Apache PIG.** Strong points of using Apache PIG are integrated workflow optimization techniques, an independence of the underlying execution platform and a large and active community of developers that maintain and expand the project. The abstraction layer was found to be flexible and easy to use for persons with a programming background, but the scripting environment may not be intuitive for all data scientists.

## 4.2 Integrated Approach

Based on these experiments, we propose to marry both approaches to create a best-of-both-worlds approach, in which we combine the strengths of Taverna and Apache PIG. We propose to extend the PPL-translator to convert workflows defined in Taverna into PIG scripts which are then compiled to optimized MapReduce programs. We thereby gain access to a wealth of work that is being done by PIG's active community, especially with regards to supported workflow elements and optimized workflow execution. In addition, with the emergence of PIG compilers to other parallel processing paradigms, Taverna users can create components that may be run on other frameworks than MapReduce such as Apache Spark.

This proposed solution allows us to model workflows that contain dataflow elements such as Joins and handle multiple inputs and outputs, extending the reach of Taverna dataflow elements that we are able to compile to MapReduce with the PPL-translator. Taverna users can create workflow components that can be either be run directly on MapReduce, or be embedded in more complex workflows using workflow management systems.

For more complex dataflows that contain dataflow elements outside of what Apache PIG can model, the SCAPE PT subproject is investigating the use of Apache Oozie [IHB+12]. Oozie provides a workflow engine that allows one to chain different types of Hadoop-based programs together, such as MapReduce applications, PIG scripts, or sequential Java programs. It also provides a REST interface for submitting (and scheduling) jobs on Hadoop clusters. In this framework, Taverna can be used to create workflow elements that are compiled to PIG scripts or MapReduce jobs using the PPL-translator and then run either as stand-alone jobs or as components in more complex workflows using the Oozie interface.

## 4.3 Open Questions

One open question we are currently investigating is the problem of how to effectively execute SCAPE components as UDFs in preservation workflows. The currently proposed solution, i.e. the invocation of local tools that need to be installed on an entire cluster of machines is possible, but somewhat suboptimal in terms of setup costs and maintenance as this may impact the usefulness of parallel



preservation workflows. Together with the SCAPE community we aim to discuss methods of defining UDFs in ways that facilitate execution on a larger cluster of machines.

## 5 Conclusion

In this deliverable, we presented two different methods for parallelizing preservation workflows and discussed advantages and disadvantages of both approaches.

The first approach, compiling Taverna workflows directly to MapReduce, has the advantages of providing data scientists with a graphical interface and a large user community, but has the drawbacks that complex workflows require substantial effort and some programming knowledge to define, as well as that Taverna offers dataflow elements that cannot be effectively parallelized. In addition, directly compiling Taverna to MapReduce creates a dependence on this paradigm, which is only one of several emerging paradigms in cloud computing.

The second approach, defining workflows in Apache Pig, has the advantages that complex workflows can be defined in a scripting environment, which are both optimized and independent of the underlying execution platform. There is also a large and active community of developers that maintain and expand the project. However, the scripting environment may not be intuitive to preservation scientists.

Based on our findings, we proposed a solution that combines the strengths of both approaches. We define preservation workflows that can be both inspected with Taverna and automatically compiled to Apache PIG, which in turn executes on MapReduce. This allows us to process the scope of dataflow elements that preservation scientists require, and effectively enable parallel execution.

## 6 References

[AKM13] Alan Akbik, Oresti Konomi, and Michail Melnikov. Propminer: A workflow for interactive information extraction and exploration using dependency trees. In *ACL System Demonstrations*. Association for Computational Linguistics, 2013.

[AL12] Alan Akbik and Alexander Löser. Kraken: N-ary facts in open information extraction. In *AKBC-WEKEX*, pages 52-56. Association for Computational Linguistics, 2012.

[AVH+12] A. Akbik, L. Visengeriyeva, P. Herger, H. Hemsén, and A. Löser. Unsupervised discovery of relations and discriminative extraction patterns. In *Proceedings of the 24th International Conference on Computational Linguistics*, 2012.

[AVKL13] A. Akbik, L. Visengeriyeva, J. Kirschnick, and A. Löser. Effective selectional restrictions for unsupervised relation extraction. In *Proceedings of the 6th International Joint Conference on Natural Language Processing*, 2013.

[Bor07] Dhruba Borthakur. *The Hadoop Distributed File System: Architecture and design*, 2007.

[CCS09] Charles L Clarke, Nick Craswell, and Ian Soboro. Overview of the trec 2009 web track. Technical report, DTIC Document, 2009.

[CSC11] Gordon V Cormack, Mark D Smucker, and Charles LA Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval*, 14(5):441-465, 2011.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107-113, 2008.

[EST+13] Stephan Ewen, Sebastian Schelter, Kostas Tzoumas, Daniel Warneke, and Volker Markl. Iterative parallel data processing with stratosphere: An inside look. *SIGMOD*, 2013.

[FAS+13] Luis Faria, Alan Akbik, Barbara Sierman, Marcel Ras, Miguel Ferreira, and Jose Carlos Ramalho. Automatic preservation watch using information extraction on the web: a case study on semantic extraction of natural language for digital preservation. 2013.

[IHB+12] Mohammad Islam, Angelo K Huang, Mohamed Battisha, Michelle Chiang, Santhosh Srinivasan, Craig Peters, Andreas Neumann, and Alejandro Abdelnur. Oozie: towards a scalable workflow management system for Hadoop. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, page 4. ACM, 2012.

[KVB13] Vasiliki Kalavri, Vladimir Vlassov, and Per Brand. Ponic: Using stratosphere to speed up pig analytics. In *Euro-Par 2013 Parallel Processing*, pages 279-290. Springer, 2013.

[LAS+13] Marcus Leich, Jochen Adamek, Moritz Schubotz, Arvid Heise, Astrid Rheinlander, and Volker Markl. Applying stratosphere for big data analytics. In *BTW*, pages 507-510, 2013.

[LLC+12] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with MapReduce: a survey. *ACM SIGMOD Record*, 40(4):11-20, 2012.

[OAF+04] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045-3054, 2004.

[ORS+08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099-1110. ACM, 2008.

[TSJ+09] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wycko, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626-1629, 2009.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, pages 10-10, 2010.