Constructing Scalable Data-Flows on Hadoop with Legacy Components

Rainer Schmidt, Matthias Rella, and Sven Schlarb
AIT Austrian Institute of Technology GmbH
Donau-City-Strasse 1, 1220 Vienna, Austria
rainer.schmidt@ait.ac.at

***Abstract*—In recent years, data intensive computing platforms have gained wide spread adoption in diverse domains. The Apache Hadoop ecosystem provides a rich software stack that supports data analysts in managing large volumes of data at different levels of abstraction. Higher level frameworks provide support for crafting scalable data-flow applications, creating databases from unstructured data, and for performing business analytics tasks. We have, however, also observed a strong need for the integration of existing software tools into data-intensive applications in order to carry out data and/or application specific tasks that are not directly supported by the platform or that cannot be re-implemented with reasonable effort. Examples are the processing of multimodal data (image, video, audio), the integration of scientific, often community-specific, libraries, or the execution of complex and experimental data mining algorithms. We present ToMaR, a flexible application for integrating existing software into data-flow applications that execute on top of a MapReduce-based environment. The application supports a Linux-inspired pipes-and-filter based syntax, the execution of existing applications using file and stream based IO, and the efficient integration with existing data-flow frameworks like Apache Pig. The paper describes general techniques implemented and used by ToMaR and their application in context of the EU project SCAPE.**

I. Introduction

Driven by the need of processing data volumes at the Internet-scale, scale-out solutions for storing and analyzing large volumes of content using large clusters of commodity computers have been developed [1]. Data-intensive computing frameworks typically provide a distributed files system, as well as an efficient execution environment which is supported by a parallel programming model and execution strategy. While these frameworks where initially targeting the implementation of scalable applications for web indexing, log file analysis, or graph processing, recent developments have shown a much broader applicability of data-driven solutions. Higher-level services implemented on top of the frameworks, enable users to handle large volumes of unstructured data based on higher-level languages enabling one for example to build up distributed databases and to utilize business intelligence tools [2] [3].

Apache Hadoop provides an open-source implementation of the MapReduce framework introduced by Google in 2008 [4]. In recent years, a vibrant community as well as a rich eco-system of Hadoop-related projects evolved supporting scalable and distributed data-processing applications at various levels. Projects like Apache Pig or Hive enable a direct mapping of data management techniques like commonly database operations to massively parallel architectures allowing one to perform well known data analytics tasks on large-scale data sets by using higher-level data analytics frameworks [5]. Next-generation "Big Data" frameworks like Apache Spark and Flink introduce new architectures, richer programming models and higher-level languages [6].

Parallel data processing analytics environments, however, require the user to conform to a parallel programming model (like MapReduce), take advantage of a higher-level language (like Pig or Hive), or make use of utilities like the Hadoop streaming API. While these methods provide key abstractions for processing large-scale data sets, it is still required to integrate existing software components into the data processing chain. The loading of complex data formats like raw scientific data, encoded video or audio data typically requires the utilization of specific libraries that are able to decode and process the data. In many domains, these tools are however only available in the form of existing and highly optimized libraries and stand-alone applications. In these cases it is clearly not feasible to re-engineer individual application components to fit a particular programming model but it is nevertheless often desirable to take advantage of data-intensive computing platforms.

We present ToMaR, an application that provides flexible and easy to use mechanisms for executing legacy applications on Hadoop clusters. ToMaR is intended to enable users to execute existing applications on the cluster in a similar way they are executed in a desktop environment. ToMaR is extensible regarding the used application invocation mechanisms, workflow and execution engines, as well as the utilized storage systems (like HDFS or Microsoft HDInsight blob storage). In the context of the SCAPE project, ToMaR has been evaluated based on three different application areas including data sets such as high-resolution images from digitized books, Web archiving snapshots, and large-volumes of data items generated by scientific instruments.

This paper describes the basic concepts used and implemented in ToMaR. This includes the strategies for handling application invocation and IO, application composition, and the handling data locality for referenced content. We report use-cases from the SCAPE project and conclude with a brief evaluation of ToMaR's support for executing scalable data flows on top of the Apache Pig data analytics platform.

II. Legacy Application Support

Implementing custom MapReduce applications provides a powerful way for creating robust and scalable applications that

are IO bound. For many domains, it is however not possible to implement the entire data acquisition and analysis workflow based on Java, MapReduce, and database abstractions. Porting existing image processing tools to native Hadoop applications would not be feasible in many cases. The fact that data-intensive computing frameworks are required to handle data sets at a certain scale motivates the development of suitable approaches for integrating existing and domain-specific applications on these platforms.

A. Limitations of Built-in Mechanisms

The invocation mechanism and IO handling for legacy applications is governed by a specific tool descriptor when using ToMaR. Hadoop has a built in mechanism that supports the execution of scripts like Bash or Python which are automatically translated and executed as MapReduce applications. This mechanism provides convenient support for processing data with common UNIX filters but is limited with respect to integrating arbitrary applications. Examples are applications that do not support IO steaming or operations that are requiring multiple inputs (which might be passed by reference). Hadoop is designed to process its input based on key/value pairs. This means the input data is interpreted and split by the framework, which makes it also difficult for processing binary data. Legacy applications in general do not support HDFS file pointers and often do not support IO streaming through stdin/sdout making these tools difficult to use on Hadoop in general.

B. Application Examples

ToMaR is a generic MapReduce application that has been developed in SCAPE to provide efficient legacy application integration for scalable content processing. The development of ToMaR in the SCAPE project was motivated by a strong need for executing tools and workflows in a scalable fashion and to deal with data volumes beyond test examples. This was hindered by the diversity of applications developed and used by researchers. Data-flow applications where often implemented using desktop workflow environments, scripts chaining together multiple command-line tools, or using algorithms implemented in packages like Matlab. Example applications involved the calculation, extraction and comparison of metadata, detection of the file formats or codecs, and transcoding or signal processing. Matchbox [7], Jpylyzer [8], Pagealizer [9], or xcorrSound [10] are examples for tools that have been developed in the context of SCAPE, each addressing different preservation activities and types of content. Examples scenarios from the SCAPE project are:

- The detection of image duplicates and cropping error in millions of scanned book pages, using Matchbox as a tool for image comparison using computer vision approach based on the OpenCV library.
- Preserving scientific data by conversions from the RAW format to NeXus which is a proton and neutron community format, using community tools and HDF5 libraries.
- Deep characterization of Web archives for millions of resources, using different format validators and metadata extractors (formats, encodings, validity).
- Quality Assurance for archived content involving audiovisual content, using applications for the comparison of images (like browser snapshots) or audio files (through cross correlation).

A common characteristics of the employed workflows was their resource intensiveness often caused by the processing of large volumes of binary input files. Besides specific calculations, third party tools where often used for generating large data sets from binary content. This process step can be easily exectued in a data parallel fashion and the resulting data sets could be subsequently processing very well using common data management services available on systems like Hadoop. Due to the large variety and complexity of these tools, it was however neither feasible to restrict the utilized thrid party applications by language, supported IO mechanisms or API, nor to reimplement their functionality to fit a particular execution environment.

III. Basic Concepts and Mechanisms

A. Specification Language Features

ToMaR relies on the SCAPE Tool Specification Language which provides a simple XML schema[1] for formalizing the usage of different software components (tools) by specifying properties like API calls, configuration parameters, or defining how IO is handled. Tool specification documents (called toolspecs) are XML documents which define a set of operations that can be carried out by the tool they define. Tool specification documents can describe general patterns for using a single software package or define new operations. The operations defined in the toolspec documents specify atomic operations that can be carried out at scale using ToMaR. A tool specification document comprises a set of fields that provide general information including author, licensing, and installation requirements. The *operations* element defines one or more atomic operations, consisting of a textual description, a definition of the software to invoke as well as a specification of inputs and outputs. The *inputs* and *outputs* elements provide a definition for each input and output including unique names and a definition how IO is handled (e.g. based on files or std. streams). Tool specification documents also provide a simple and flexible mechanism to define tool dependencies, in particular for workflows. Applications that have been developed in the context of SCAPE are provided as Debian packages. These packages come with a toolspec document by default which can consequently be resolved using Linux package management.

B. Control File

If running as a standalone MapReduce application, ToMaR takes a generated plain text file as input (called a control file) specifying how one or multiple joined toolspec operations are applied to the payload data. Each line in the input file (called a control line) typically applies the specified command to a particular data item (e.g. a file). Hence, a user could generate a control file with n lines in order to process n input files with ToMaR. The control file is broken into splits at execution time and processed independently by distributed mapper processes. When applied together with a higher-level platform

[1] available at https://github.com/openplanets/scape-toolspecs/blob/master/toolspec.xsd

like Apache Pig, the input for ToMaR can be dynamically generated and it is no longer required to create a control file before execution time. However, the syntax for specifying the commands to be executed by ToMaR on the cluster are similar, regardless if the control lines are generated statically or on-the-fly. In the following we give a short overview of the control file specification presently supported by ToMaR. For more detailed information the reader is referred to the ToMaR documentation on Github[2].

A control line specifies the invocation logic for one or multiple operation defined by a tool specification document including concrete values for parameters (like file references). At present, ToMaR supports references to the HDFS file system as well as the Azure blob storage (WASB) as a distributed data source / data sink. The tool specification language also supports the definition of commands that read from standard input and/or write to standard output, which can be consequently used for specifying a control line that invokes an operation that streams data from/to a file on HDFS indicated by the '>' character. Instead of streaming a command's output to a file, the output stream can also be redirected to another toolspec command, similar to using pipes on the UNIX shell. Compared to performing these tasks separately, the mechanism allows one to create chained operations which are handled by ToMaR within a single map invocation step. IO redirection between toolspec operations is indicated with the 'l' character. ToMaR supports application shipping for Java-based applications relieving ToMaR from creating expensive OS-level processes for invoking legacy applications, which can significantly improve the application performance.

IV. Data Decomposition

A. Splitting Input Data

The control file provides the input data of the MapReduce application, which is split and distributed across the cluster nodes. Hadoop creates an input split for each line by default (causing the creation of a map task for each line of the control file). In most cases it is therefore advisable to configure the number of lines per split (-n option) with respect to the cluster size as well as the nature of the application. The total number of splits for an input file might be selected to be a fraction of the maximal available map tasks. There is however a trade-off between large splits (less efficient with respect to fail-over/load balancing) and small splits which increase the overhead on application. Each line of a split (called a record) is processed iteratively on the worker node the split has been assigned to by the framework. The workload that is imposed on a node by a single record depends on the formulation of the control lines, which can significantly vary between different applications.

B. Handling Data Locality

While Hadoop is capable of exploiting data locality by allocating CPUs for processing the splits of a large input file which reside closely to the data, this mechanism fails for input files containing only references to the payload data. Although the control file holds information on data locality implicitly it is not exploited per se. ToMaR implements a specific input format class called ControlLineInputFormat which enables Hadoop to exploit data locality for the distributed processing of referenced content in control files. This is achieved by sorting and splitting the control file with respect to the referenced data blocks. Our algorithm identifies the required data blocks (and their replications on HDFS) for each record. After generating a sorted list of hosts for each control line, a new control file that is sorted with respect to data locality is generated. Using the newly generated input file, ToMaR is capable of creating sorted data splits which carry information regarding their location, which in turn can be exploited by the Hadoop execution environment as shown in Figure 1.

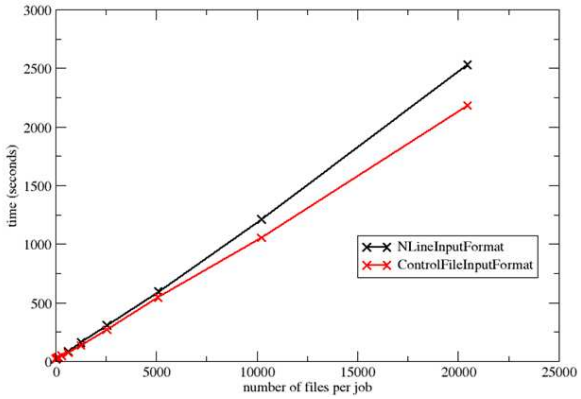

Fig. 1: Performance results for calculating MD5 hashes for referenced content. The ControlFileInputFormat provides support for handling data locality based on sorted and annotated input files.

V. Creating Complex Data Flows

A. Higher-level Data-flow Language

Higher-level languages like Apache Pig support the development of complex data-flow applications for large volumes of data as for example provided by Extract, Transform, Load (ETL) processes. Pig provides a convenient data flow language allowing users to efficiently perform data transformations using database abstractions. Pig's concept of user-defined functions (UDFs) enables one to develop custom functionality in Java. UDFs are registered with a Pig Latin script and used as part of a statement for example to load/store, filter, or aggregate data. These mechanisms are also well suited to integrate complex software components like required for performing machine learning tasks [11].

B. Data Extraction Example

In the following we present a fraction of a larger data flow application which was implemented to analyze different versions of images resulting from large-scale book scans in the SCAPE project. The workflow has been implemented as a Pig Latin script and makes use of ToMaR to invoke FITS[3], a software component that identifies, validates, and extracts technical metadata for various file formats. FITS wraps several third-party open source tools, normalizes and consolidates their output. The snippets in figure 2 shows examples for different operations defined in the FITS tool specification document.

[2]https://github.com/openplanets/tomar

[3]https://code.google.com/p/fits/

FITS is applied on a set of files residing on HDFS in order to generate XML metadata. The used operation "j-identify" makes use of Java shipping to invoke FITS by ToMaR within its JVM (as compared to using the command-line interface). Data is read from the file system and the output stream is redirected to the next operation "j-xpath-stdin", which evaluates an XPath expression and returns the result to stdout. The example snippet in figure 3 shows a control file that makes use of two chained toolspec operations.

```
<operation name="identify">
 <command>/bin/fits.sh -i ${input} -o ${output}</command>
...
<operation name="identify-stdout">
  <command>/bin/fits.sh -i ${input}</command>
...
<operation name="j-identify-stdout">
  <command>/usr/bin/java
     edu.harvard.hul.ois.fits.Fits -i ${input}</command>
```

Fig. 2: Different operations defined within the FITS tool specification document.

```
fits identify-stdout --input="hdfs:///extracted.10" |
  xpath j-xpath-stdin --xpath="//identity[1]/@mimetype"
fits identify-stdout --input="hdfs:///extracted.11" |
  xpath j-xpath-stdin --xpath="//identity[1]/@mimetype"
...
```

Fig. 3: Control file for extracting mime type information using two joined operations.

ToMaR implements the necessary hooks to be registered and used within a Pig Latin script. This is motivated by the fact that ToMaR is typically used as part of a larger workflow that needs to analyze the data produced by third party tools. By utilizing ToMaR within a Pig script, one can directly pick up results generated by legacy applications and process and analyze them on Hadoop. Here, data is directly loaded from/into Pig relations and there is no need to generate a control file prior to the execution. The generation of map and reduce tasks for performing data manipulations are entirely delegated to the Pig platform. The example snippet in figure 4 shows a Pig latin script that invokes FITS using ToMaR.

```
REGISTER tomar-1.4.2-SNAPSHOT.jar;
DEFINE ToMarService eu.scape_project.pt.udf.ControlLineUDF();
DEFINE XPathService eu.scape_project.pt.udf.XPathFunction();

pth = LOAD '$image_paths' USING PigStorage() AS
  (image_path: chararray);

fits = FOREACH pth GENERATE image_path as image_path,
  ToMarService('$toolspecs',
    CONCAT(CONCAT('fits stdxml --input="hdfs://', image_path), '"'))
  as xml_text;

mimes = FOREACH fits GENERATE image_path, XPathService
  ('$xpath_exp1', xml_text) AS node_list1;
```

Fig. 4: Pig Latin snippet using ToMar and XPath based on UDFs. Resulting data sets are loaded into a Pig relations.

Experimental tests showed no significant impact on the performance if ToMaR is utilized on top of the Pig platform as compared to being run as a standalone MapReduce application. An experiment in which 50.000 random files have been processed in 1000 splits on a 10 node development cluster led to the results shown in table 1. The above described workflow fragment has been executed in Pig Latin as well as using ToMaR as standalone MapReduce applications. Both workflows have been executed with and without using ToMaR's support for Java application shipping.

Application/Invocation	Java	Command line
ToMaR (standalone)	4.21	9.02
ToMaR with Pig	4.42	9.32

TABLE I: Execution time in hours for processing 50.000 random files with the tools FITS and XPATH using ToMaR as standalone MapReduce applications and in combination with the Apache Pig platform. Tool invocation was performed directly in Java and alternatively via the command line interface.

VI. Conclusion

We have presented an approach and application for integrating existing applications into scalable data flows. ToMaR provides a flexible application that has been developed and evaluated in the context of the EU project SCAPE for analyzing large volumes of content, originating from different domains, on scalable environments like Apache Hadoop. In this paper, we describe a set of mechanisms that support the flexible and efficient integration of third party applications in a MapReduce environment. We give an example how application logic can be implemented in ToMaR and executed as part of arbitrary MapReduce based workflows. Furthermore, we show that the presented approach can also be efficiently integrated with higher-level services like provided by the Apache Pig framework. In this setting, ToMaR is solely used for application integration while operations like data representation, decomposition, and/or processing - which are specific to the MapReduce execution environment - are entirely delegated to the data analytics platform.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proc. of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. NY, USA: ACM, 2003, pp. 29–43.

[2] A. F. Gates, O. Natkovich *et al.*, "Building a high-level dataflow system on top of map-reduce: The pig experience," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1414–1425, Aug. 2009.

[3] A. Thusoo, J. S. Sarma *et al.*, "Hive - a petabyte scale data warehouse using Hadoop," in *ICDE '10: Proc. 26th Int. Conference on Data Engineering*. IEEE, Mar. 2010, pp. 996–1005.

[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008.

[5] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: A survey," *SIGMOD Rec.*, vol. 40, no. 4, pp. 11–20, Jan. 2012.

[6] V. Markl, "Breaking the chains: On declarative data analysis and data independence in the big data era," *Proc. VLDB Endow.*, vol. 7, no. 13, Aug. 2014.

[7] R. Huber-Mörk and A. Schindler, "An image based approach for content analysis in document collections," in *Advances in Visual Computing*, ser. Lecture Notes in Computer Science, 2013, vol. 8034, pp. 278–287.

[8] van der Knijff *et al.*, "Improved validation and feature extraction for jp2 images: the jpylyzer tool," in *Archiving Conference*, vol. 2012, no. 1. Society for Imaging Science and Technology, 2012, pp. 264–269.

[9] A. Sanoja, S. Gançarski *et al.*, "Block-o-matic: a web page segmentation tool and its evaluation," *BDA, Nantes, France*, 2013.

[10] B. A. Jurik and J. A. S. Nielsen, "Audio quality assurance: An application of cross correlation," *Proc. IPRES*, 2012.

[11] J. Lin and A. Kolcz, "Large-scale machine learning at twitter," in *Proc. of the 2012 ACM SIGMOD Int. Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012.