Enhancing VLAM Workflow Model with MapReduce Operations

Mikolaj Baranowski[1], Adam Belloum[1] and Marian Bubak[1,2]
[1]*Informatics Institute, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands*
[2]*AGH University of Science and Technology, Department of Computer Science, Mickiewicza 30, 30-059 Krakow, Poland*

Keywords: Scientific Workflow, Ruby, MapReduce, Hadoop, Parallel Processing.

Abstract: MapReduce frameworks proved to be a good solution for storing and processing large amounts of data. Thanks to data parallelism, they allow to move computations very close to the storage and therefore to reduce an influence of "I/O bottleneck". Workflow Management Systems, in turn, are widely used for modeling of scientific applications. Users that are willing to use MapReduce frameworks in their workflows have to run separate environment to develop Map/Reduce operations. In this paper we propose an approach that will allow to extend existing application models by MapReduce routines. Our solution bases on DSL constructed on top of Ruby programming language. It follows examples of Sawzall and Pig Latin languages and allows to define Map/Reduce operations in minimalist way. Moreover, because the language is based on Ruby, the model allows to use user defined routines and existing Ruby libraries. A particular model of the workflow management system can be extended with our DSL letting users to use one environment for developing the workflow and MapReduce application.

1 INTRODUCTION

The term data-intensive applications refers to a computer software that process large volumes of data. Recently, we observe an explosion of data and clearly, these kind of applications are going to play more important role in science. As authors of (Hey et al., 2009) point out today's computers have relatively low I/O performance. It implies that algorithms has to be designed in a different way and should follow the rule which says that the computation should be performed as close to the place where data is stored – in a sense of time distance – as possible. Building larger, wider clusters and data centers will not solve the issue of "I/O bottleneck".

MapReduce is a programming model that has abilities of processing large volumes of data since the computations are moved as close to data storage as possible. On the other hand, Workflow Management Systems proved to be an easy and efficient way of describing complex systems and being adaptable to use new technologies. In (Goble and Roure, 2009), authors define the workflow as a "precise description of a scientific procedure – a multi-step process to coordinate multiple tasks, acting like a sophisticated script". As a task they consider "a running program, submitting a query to database, submitting a job to a compute cloud or grid or invoking a service over the Web to use a remote resource". Clearly, workflows have to find their place in a data-centric research as a tool for efficient and easy orchestration of tasks.

There are many ways of implementing Domain Specific Languages. Newly designed language can be created together with parsers and interpreters but also it can be built on top of an existing one. Ruby programming language has special abilities for this kind of purposes. It proved to be very malleable in many projects based on DSLs such as Rake, Cucumber or Sinatra. It has a good reputation as a language for designing DSLs also among users of other technologies (Ford, 2013).

The main objective of our work is to unify models of MapReduce and Workflow Management System (WMS) to provide one environment that will allow users to define their computations in efficient way. If there is a need to use data stored in a MapReduce-oriented storage like Hadoop Distributed File System (HDFS), one has to develop two applications, the one that gains data from a storage (MapReduce) and the second one that processes the data (workflow). Conventionally, they have to be defined using two different environments. We intend to simplify this process by defining MapReduce operations and a workflow in one model. Moreover, we would like to propose a solution which would not resign from MapReduce elegance and would follow MapReduce origins of func-

Baranowski M., Belloum A. and Bubak M..
Enhancing VLAM Workflow Model with MapReduce Operations.
DOI: 10.5220/0004488401790185
In *Proceedings of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2013), pages 179-185
ISBN: 978-989-8565-69-3

tional programming and their valuable features such as minimalism. The main contribution of this paper is an efficient approach to combine a workflow model with a MapReduce model to provide a complete solution that would gain data from MapReduce resources and then consume them in a workflow.

This paper is organized as follows. In section 2, we describe the MapReduce model, its implementations and MapReduce specific DSLs. Section 3 contains information about workflow systems and their integrations with MapReduce frameworks and other works related to the topic of this paper. VLAM application is described in section 4 and a design of our solution in 5, it is succeeded with a section about implementation – 6. The whole paper is completed with an example application in section 7 and section 8 which discusses future work.

2 MapReduce MODEL

The MapReduce (Dean and Ghemawat, 2008) programming model was designed for processing large datasets by Google. It is inspired by *map* and *reduce* functions from Lisp and other functional programming languages. The computation consumes a set of input key/value pairs, passes each of them to a map phase where they are transformed to intermediate key/value pairs. Then, intermediate values that are associated with the same key are grouped together and passed to reduce phase where whole set is processed and the final answer is calculated. As it takes the advantage of data parallelism, recursive data structures are impossible to process.

Beneath, there is a description of Hadoop framework – as an example of MapReduce implementation – and its most important features that are used in our work. Domain Specific Languages created for MapReduce model, such as Pig (Olston et al., 2008) Latin (woks in Hadoop environment) and Sawzall (Pike et al., 2005) are also mentioned.

Hadoop is an open source MapReduce framework inspired by Google's work on MapReduce and Google File System. It is implemented in Java programming language together with closely related applications such as Hadoop Distributed File System. Map/Reduce operations are defined in Java programming language as classes that implement a required interface.

Pig Latin (Olston et al., 2008) is a hybrid SQL-like declarative language with MapReduce approach. It was created by Yahoo Research team for Pig application that is built on top of Hadoop in a purpose of providing easier interface for MapReduce processing. Map/Reduce operation are compiled from the SQL-like statements that use special commands such as `LOAD` for loading data set, `FILTER` for specifying filtering condition or `GROUP` for grouping records. The important feature of Pig environment is a support for User-defined functions (UDFs) – currently they can be written in Java and used in any construct including `FILTER` or `GROUP`. Sawzall (Pike et al., 2005) in turn, was created by Google to process large number of log records with a MapReduce methodology. The name Sawzall refers to a whole MapReduce environment and, in particular, to the programming language which is used to describe Map operations. It is a statically typed language that is compiled to a machine code. It supports complex data types like lists, maps and structures. Two fundamental features of this tool are as follows: operations can be performed only on single records and the result of an operation is returned by `emit` statement which sends data to an external aggregator. Aggregators are implemented in C++ to gain the maximum possible efficiency. They can occupy thousands lines of code describing how a data flow is managed.

Existing solutions do not fulfill our needs, Sawzall is a programming language that cannot be used outside of its ecosystem, Pig Latin is also strongly tided to the Pig application. Other tools – open source oplications like MRToolkit (created by New York Times team) and Dumbo (Last.fm) are designed to use streaming interface in Hadoop, however, as we explain in the following section, introducing type specification in Map operation and using Reducers implemented in Java can lead us to the more efficient model.

3 MAPREDUCE AND WORKFLOW SYSTEMS

There are workflow systems that provide access to MapReduce solutions from a workflow model level. Because they present different approaches, the experience gained by their authors can be exploited to evaluate our concepts.

The Kepler Project (Ludäscher et al., 2006) is a system created for designing, executing and sharing workflow models. Applications can be constructed from entities called *actors* that represent computational components and *channels* that specifies data flow. The execution of a workflow is managed by entity called *director* that consists execution parameters and that coordinates actor execution order. In (Wang et al., 2009), authors describe how they execute MapReduce tasks from Kepler. Authors mention

Table 1: Comparison of MapReduce oriented DSLs.

Feature	Sawzall	Pig Latin
Execution environment	Sawzall	Pig / Hadoop
Programming model for Map operations	They are developed in Sawzall programming language (statically typed, compiled to machine code)	In Pig Latin (a hybrid of SQL like declarative language with MapReduce approach), map operation is compiled from a `FILTER` and `GROUP` statements
Programming model for Reduce operations	Only name should be specified in Sawzall code. Aggregator is implemented in C++ to achieve the best efficiency	Reduce phase is compiled from `GROUP` command

that they wanted to focus on a solution that would provide the universal interface to MapReduce and would not be limited to any particular domain. In fact, their solution where Map/Reduce operations are defined as separate workflows fulfills this requirement and can easily handle any data structure. In order to execute MapReduce operations on Hadoop, the MapReduce actor has to be created. It consists two sub-workflows that are responsible for two phases - the Map phase and the Reduce. Kepler execution engine and MapReduce sub-workflows are distributed to working nodes to perform MapReduce tasks (Wang et al., 2009). In the experiment authors measured execution times implementing MapReduce application using Kepler and using Java implementation – without Kepler engine. They found that Java implementation is four to six times faster than the solution based on Kepler. After further experiments they conclude that this overhead is due to Kepler engine initialization and workflow parsing. Authors do not consider using Reduce functions implemented in Java as well as Map/Reduce functions from Hadoop library that allows to perform Map/Reduce operations more effectively. It may significantly speedup execution for more complex Reduce operations and for simple one such as word count (benchmark example in (Wang et al., 2009)) it may decrease slowdown caused by Kepler initialization and workflow parsing.

MRGIS (Chen et al., 2008) (MapReduce-enabled GIS) is a workflow system with MapReduce integration for Geographical Information System (GIS). It works with Hadoop to provide a computing platform for GIS applications. Using the environment of MRGIS, users are able to define tasks using Graphical User Interface or Python scripts. Authors show that this solution solves an issue very efficiently, however it is very specific to its domain and it can not be applied to solve more generic issues.

The approach of defining workflows in programming languages is also worth to mention. In (Baranowski et al., 2013b), there is a try of transforming applications written in general purpose programming language (Ruby) into workflows. In (Thain and Moretti, 2010), authors describe an approach of describing complex applications using scripting language similar to Makefile syntax.

Hadoop framework is the only MapReduce open source application that is widely used thus it seems to be an obvious choice of MapReduce environment. There are solutions that integrate Hadoop with workflow systems, their authors had different goals – Kepler-based solution was intended to provide easy-to-use MapReduce environment for workflow users while the aim of MRGIS system was to improve a workflow execution.

4 WS-VLAM WORKFLOW MANAGEMENT SYSTEM

The goal of our work is to enrich WS-VLAM application model with MapReduce constructs to allow defining Map/Reduce operation in workflow description.

WS-VLAM is a workflow management system which covers the entire lifecycle of scientific workflows from design through execution phase to sharing and reuse complete workflows and their components (Cushing et al., 2011). As it is shown in Figure 1, the center point of VLAM architecture is a message queue. On the left side, there are modules responsible for coordinating task execution (Task Auto-Scaling, Submitter and Monitor) and on the other modules responsible for a connection with resources. Monitoring is performed at both the workflow level and workflow component levels. At the workflow level, the end user can follow the state of a workflow submission and check whether the workflow is pending, submitted, running, or completed (Belloum et al., 2011). VLAM workflow model bases on a directed graph representation.

A solution described in (Baranowski et al., 2013a)

Table 2: Comparison of integrations of workflow and MapReduce systems.

Feature	Kepler+Hadoop	MRGIS
MapReduce framework	Hadoop	Hadoop
Implementing Map/Reduce operations	Map and Reduce operations are implemented as Kepler workflows	operations are provided by the environment
Execution of Map/Reduce operations	Kepler engine has to be deployed on each worker to parse and execute Map/Reduce subworkflow	Needed operations were wrapped so they can be executed on MapReduce platform
Main goal	Easy to use	High performance

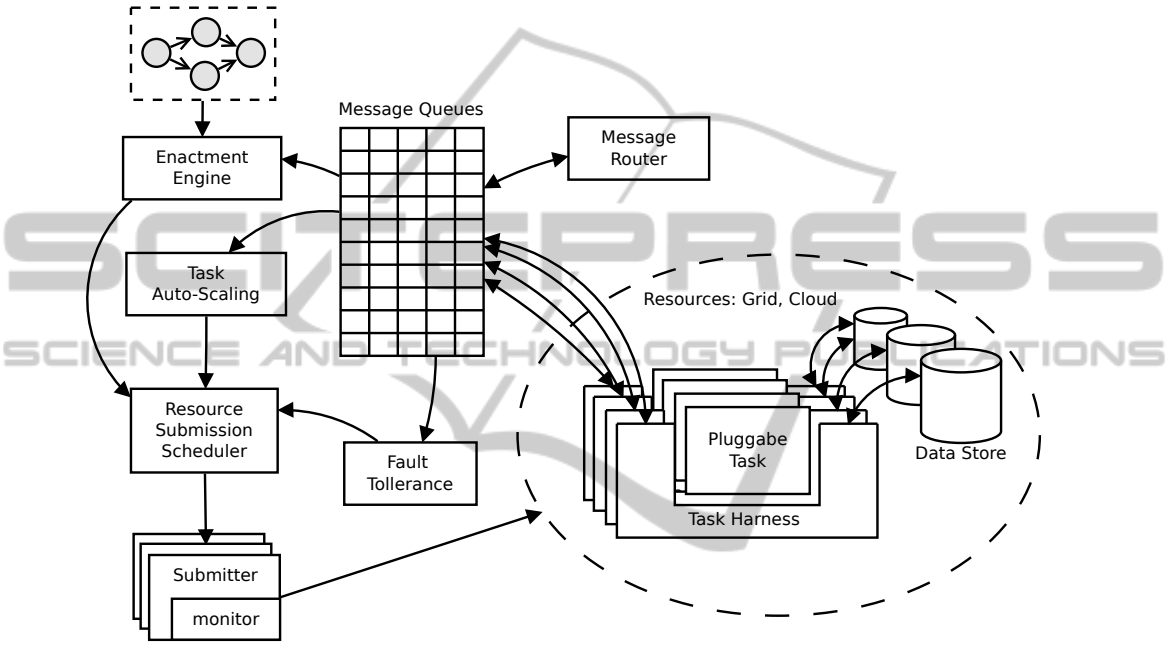

Figure 1: Architecture of VLAM.

investigates possibility of using a scripting programming language to describe MapReduce operations in WS-VLAM workflow systems and the current paper is a direct continuation of this work.

5 MapReduce DSL

We choose Hadoop as a targeting environment for our solution and Ruby programming language as a base for newly designed Domain Specific Language (DSL). There are two ways of using Hadoop environment – by Java classes and by the streaming interface that we want to focus on. Following the thought from Section 3 about Sawzall where aggregators are defined in C++, we would like to propose a language for defining Map operations and keep Reduce operations implemented in Hadoop native way – the suitable set of Reduce operation will be provided. To argue the chosen approach, we would like to refer to (Pike et al., 2005) where authors count out arguments:

- Map operations are frequently changed during developing process and every often written from scratch,
- most of the time spent in the execution of Map operations is spent on waiting for I/O events, not on computations itself,
- users use a small set of Reduce operations,
- reduce operations have big influence on overall performance.

Thus, it is important to provide environment in which Map operation is implemented in a way that ensures cheap maintenance and modifications. On the other hand, the map/aggregate phase is implemented using a technology that provide maximum efficiency – possibly the tool's native technology and, a set of the most commonly used reduce/aggregate function is provided to release a user from this duty and to achieve better robustness thanks to code reuse.

To define the Map operation we should pass a block to the function named `map`. Block takes two

Listing 1: The example of Map function defined using DSL.

```
map do |c, v|
  [c.string(v[0]), c.d_number(1)]
end

reduce(:max)
```

arguments, the first one (variable `c` in Listing 1) provides helper functions that can specify a type of the returning data. Basing on that information, our application is able to select proper Aggregator – e.g. if the user selects `Max` as a Reduce operation, depending on the type of returned data (string or long), `LongValueMax` or `StringValueMax` will be selected from Hadoop Aggregator library. Right now, user can select from three data types: `l_number` which stands for `Long`, `d_number` for `Double` and `string` data type. The second passed variable (variable `v` in Listing 1) stands for the data that is suppose to be processed in the Map phase. Designed environment does not restrict user from defining classes and functions that can be used in map operation, also Ruby libraries can be included into Map definition.

To define a new Reduce operation, developer has to specify its name – as in the last line in listing 1, the execution of MapReduce operations relies only on recognizing a relation between data types returned by Map and the name of Reduce operation.

6 IMPLEMENTATION OF MapReduce DSL

Apache Hadoop was chosen as a MapReduce framework. To control its execution, we choose JRuby (Ruby implementation based on Java Virtual Machine). Thanks to that, constructed Domain Specific Language (DSL) possess similar advantages as mentioned Sawzall and Pig Latin languages.

6.1 Implementation of Map Operation

Similar approaches based on scripting language based solution were used in commercial tools such as MR-Toolkit created by New Your Times which uses Ruby programming language and Dumbo framework developed by Last.fm which in turn is made with Python programming language.

A routine which specifies Map operation – block passed to the function `map` in Listing 1 is interpreted by JRuby as Map operation. Because it specifies data types, returned values are cast to corresponding Java data types.

6.2 Implementation of Reduce Operation

As it was mentioned before, Reduce operations is implemented in a native way. A set of already implemented operations are provided and action of a workflow user will be limited to selecting a desired operation from a list. Knowing the data type (they are specified in DSL) will let to use Hadoop Aggregate Package. For example, if a user in the Result phase wanted to sum records returned by the map function from Listing 1, `LongValueSum` would be selected as the Aggregator.

7 EXAMPLE APPLICATION

Word counter application became a standard example application for MapReduce frameworks. Map operation splits a text or texts into words and for each word it emits a pair `[<word>, 1]` where `<word>` stands for one word from the text. Then, in Reduce operation these records are grouped basing on `<word>` and all values are added. The user is suppose to choose `Sum` as a Reduce operation. Basing on that information, our application is able to select `LongValueSum` Aggregator from Hadoop library as the adequate Reduce operation. Because there are as many records as word occurrences in a text, the sum stands for a total number of occurrences of a particular word.

The implementation of the Map operation is shown in Listing 2, it splits the line of the text and emits the list of variable `res`. The last line of the listing specifies reduce operation which is `sum`.

The word count application was used to test elaborated approach. Tests were performed in DAS-4 environment that consists 8 nodes with Hadoop installed. Each node has dual quad-core 2.4 GHz CPU and 24 GB memory each connected with InfiniBand and Gigabit Ethernet. We prepared 2.6 GB of English books in a plain text format taken from Project Gutenberg. They were stored in HDFS and used in tests as a whole set and also in smaller pieces. Texts were processed and results that were produced were correct.

8 SUMMARY AND FUTURE WORK

The elaborated approach implements the environment for defining MapReduce queries in workflow models. It follows the examples Pig or Sawzall applications where Map/Reduce operations can be defined using

Listing 2: Map operation of word counter application.

```
map do |c, v|
  res = []
  v.split.each do |i|
    res << [c.string i, c.number(1)]
  end
  res
end

reduce(:sum)
```

convenient Domain Specific Languages. However, to prove the elaborated solutions to be easy and efficient, more complex applications have to be investigated in the future and execution times has to be measured and compared with native Hadoop implementation.

Proposed solution can be extended into more generic form – as a pluggable application that can be used to extend models of other applications. In order to provide such a functionality, targeting application should implement a set of routines that will coordinate MapReduce tasks from the execution engine specific to particular application. The elaborated DSL is not more complex than existing MapReduce DSLs such as Sawzall and Pig Latin. It lets user to define a Map operation in a convenient way without resigning from such features as user-defined functions or Ruby libraries. Developed DSL does not require other libraries but standard Ruby distribution, if we add that there is an implementation of Ruby for Java Virtual Machine (JRuby), we can conclude that created application can be reasonably easily adopted to many existing solutions as a separate module run in a Ruby process or in the existing JVM instance. It can be also considered that the proposed solution can be merged with an existing DSL for the other domain.

In future work, other programming languages can be considered as an alternative to Ruby. All the languages that have features required in metaprogramming such as macro instructions, templates or that are modifiable in runtime, can be considered. Special attention should be paid to statically typed languages based on Java Virtual Machine platform such as Scala programming language (Odersky et al., 2010). These modern languages can provide good constructs for metaprogramming approach and at the same time, they can directly use Java type system to allow better integration with Hadoop.

Metaprogramming approach can be also considered to describe other features of Workflow Management Systems. It can be used to enrich workflow models with a configuration of resources or security policies.

ACKNOWLEDGEMENTS

This work was partially supported by the Dutch national program COMMITand KI IET AGH grant. We would like to thank Reginald Cushing and Spiros Koulouzis from University of Amsterdam for discussions and suggestions.

REFERENCES

Baranowski, M., Belloum, A., and Bubak, M. (2013a). Defining and running mapreduce operations with ws-vlam workflow management system. In *ICCS*.

Baranowski, M., Belloum, A., Bubak, M., and Malawski, M. (2013b). Constructing workflows from script applications. *to be published in Scientific Programming*.

Belloum, A., Inda, M., Vasunin, D., Korkhov, V., Zhao, Z., Rauwerda, H., Breit, T., Bubak, M., and Hertzberger, L. (2011). Collaborative e-science experiments and scientific workflows. *Internet Computing, IEEE*, 15(4):39–47.

Chen, Q., Wang, L., and Shang, Z. (2008). Mrgis: A mapreduce-enabled high performance workflow system for gis. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 646 –651.

Cushing, R., Koulouzis, S., Belloum, A., and Bubak, M. (2011). Prediction-based auto-scaling of scientific workflows. In *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*, page 1. ACM.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Ford, N. (2013). Functional thinking: Why functional programming is on the rise. Technical report, IBM.

Goble, C. and Roure, D. D. (2009). The impact of workflow tools on data-centric research. In *Data Intensive Computing: The Fourth Paradigm of Scientific Discovery*.

Hey, A., Tansley, S., and Tolle, K. (2009). *The fourth paradigm: data-intensive scientific discovery*. Microsoft Research Redmond, WA.

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M. B., Lee, E. A., Tao, J., and Zhao, Y. (2006). Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065.

Odersky, M., Spoon, L., and Venners, B. (2010). *Programming in Scala, second edition*. Artima Series. Artima, Incorporated.

Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008). Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1099–1110, New York, NY, USA. ACM.

Pike, R., Dorward, S., Griesemer, R., and Quinlan, S. (2005). Interpreting the data: Parallel analysis with

sawzall. *Scientific Programming Journal*, 13:277–298.

Thain, D. and Moretti, C. (2010). *Abstractions for Cloud Computing with Condor*, pages 153–171. CRC Press.

Wang, J., Crawl, D., and Altintas, I. (2009). Kepler + hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, pages 12:1–12:8, New York, NY, USA. ACM.