A Workflow Description Language to Orchestrate Multi-Lingual Resources

Rui Brito and José João Almeida

University of Minho
Department of Informatics
{ruibrito,jj}@di.uminho.pt

Abstract

Texts aligned alongside their translation, or Parallel Corpora, are a very widely used resource in Computational Linguistics. Processing these resources, however, is a very intensive, time consuming task, which makes it a suitable case study for High Performance Computing (HPC).

HPC underwent several recent changes, with the evolution of Heterogeneous Platforms, where multiple devices with different architectures are able to share workload to increase performance.

Several frameworks/toolkits have been under development, in various fields, to aid the programmer in extracting more performance from these platforms. Either by dynamically scheduling the workload across the available resources or by exploring the opportunities for parallelism. However, there is no toolkit targeted at Computational Linguistics, more specifically, Parallel Corpora processing. Parallel Corpora processing can be a very time consuming task, and the field could definitely use a toolkit which aids the programmer in achieving not only better performance, but also a convenient and expressive way of specifying tasks and their dependencies.

1998 ACM Subject Classification D.3.4 Processors

Keywords and phrases workflow, orchestration, parallelism, domain specific languages, corpora

Digital Object Identifier 10.4230/OASIcs.SLATE.2014.77

1 Introduction

It is widely accepted that tasks can be very difficult to manage, mainly when they have many steps, often with time consuming tools, interdependent tasks and huge datasets. As the complexity of applications increases, so does the difficulty to efficiently manage the different available resources. Users have to adopt frameworks/toolkits, as a way to improve their path to achieve the desired goals. The aim of this paper is to provide a toolkit targeted at Computational Linguistics, which may also benefit other fields that require workflow management. Texts aligned alongside their translation, or parallel corpora, are considered very rich resources for machine translation, whose formats usually follow Translation Memory eXchange (TMX) or Probabilistic Translation Dictionaries (PTD). Tools to aid machine translation based on parallel corpora include text aligners (e.g. Moses [6]), morphological taggers (e.g. Apertium [4] and Freeling [8]) and dictionary generators (e.g. NATools [10]). Some of these tools are computationally intensive, taking too long to display useful results. Most data is the result of a complex task chain and ever-changing data, which makes the results difficult to predict and errors difficult to detect. This type of work is currently done either by hand or by very crude scripts, but could be significantly improved if the linguistics community could access a toolkit to create and manage the interrelated parallel corpora processing tasks and take advantage of current heterogeneous computing systems. The lack of such a tool is the main motivation to pursue this work, developing a Workflow Description Language (WDL), which is a DSL to orchestrate Multi-Lingual resources.

3rd Symposium on Languages, Applications and Technologies (SLATE'14).
Editors: Maria João Varanda Pereira, José Paulo Leal, and Alberto Simões; pp. 77–83

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Previous work of the authors on the Per-Fide [1] project gained insight on the requirements to specify and develop a toolkit to chain tasks and to handle large data sets with different data formats. The task chain usually contains many steps, from different tools, which may change frequently, making it difficult to keep track of the results being produced. The tasks involved are mostly irregular, where memory access patterns may not be predictable, with consequent penalties on execution time. This brings forth the need to develop a toolkit to describe tasks, their chaining, their dependencies, their types and with the need to provide environment details to better take advantage of current heterogeneous computing systems.

Given a set of tools, the user should be able to specify tasks to be executed and visually follow the workflow execution phases and results. This can be achieved with a flow-graph. Some work has been done in this area, particularly with the Makefile::Parallel [9] tool, which, provides a way of describing tasks and workflows to be performed in either a cluster environment or a single machine.

The expected contribution of the work described in this paper is a toolkit to better organize, manage and modify on the fly. The toolkit will also make an efficient use of the available computational resources, by dynamically scheduling the workload with high-level directives that abstract efficient implementations.

The Section 2 begins by describing the Workflow Description Language and its aims. Section 3 presents some examples of the language, as well as some use case scenarios. Finally, section 4 presents our conclusions on the work already done.

2 Workflow Description Language

The main goal of this paper is to describe an efficient toolkit to orchestrate computational linguistics tasks. The user specifies through a WDL how individual tasks are performed. The main features of the toolkit are as follows:

- The toolkit will be environment-aware, and thus, take advantage as best as possible of the computing units and systems available;
- The toolkit should also be able to revert back to its most recent stable state (if the task is aborted);
- Primitives to generate reports, diagrams and type signature should be provided;
- There should be a mode in which the user provides heuristics on how to best use the available computing units;
- The user should be able to specify mapReduce [3, 7, 2] functions to achieve certain tasks;
  - A cleaner and more elegant way of specifying tasks;
  - Efficient low-level implementations are hidden from the end user;
- The WDL should be Algebra based, providing:
  - Familiar operations such as composition, which makes the language clearer;
- The toolkit should contemplate not only complex cluster systems, but also single user systems.

Some of the generic goals of the toolkit to be developed include:

- A declarative approach;
- Separation of elemental/independent tools;
- Separation of resources/machines;
- Reuse of existing research;
- Expressive power.

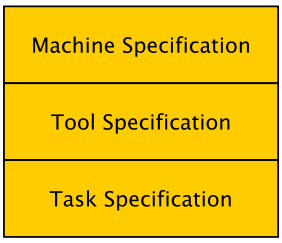

Figure 1 WDL Description.

The general organization of the WDL is described in the Figure 1. There are, essentially, three distinct parts to the language. A Machine Specification part, where all the computing systems available are described. In its simplest form, this means specifying hosts and users for remote machines. In the case of a cluster environment, there are primitives to find out which nodes are free, to ask for a full node (or nodes), since most cluster schedulers (e.g. PBS or Maui [5]) never supply the user with a full node, unless asked to. These are called execution attributes. There are also primitives to ascertain, for example, if a certain tool is available in a specific machine. Primitives of this kind are called existence attributes.

Then, there is a Tool Specification part, where all the tool to be used are described, along with any eventual execution attributes. Finally, there is an Task Specification part, where the Tasks are organized in a workflow.

This paper focus mostly on parts 2 and 3.

This WDL grammar is being developed using Perl YAPP, a Perl extension for generating and using LALR parsers. This decision is mainly because of Perl's expressive power and Regular Expressions, which make it easier to develop the kind of language we're trying to achieve.

As it can be seen from the grammar presented in Table 1, the user begins by specifying tools, which, in their most distilled form, are a collection of actions paired with an in and out type definition. The *action* production provides a way to specify how a certain tool is run. As it can be seen from the grammar, it can be either a bash script, or Perl code which is preprocessed before execution. The *execAttr* production's goal is to provide execution details for the tool, such has maximum desired walltime, whether it can run on a GPU or in Intel's MIC micro-architecture. In the case of a cluster environment, how many cores and how many nodes are needed for a specific tool. Eventually, more attributes will be added, or even changed. Once the tools are specified, the user begins specifying tasks. The production shown here (*taskSpec*), while not complete, hints at the direction we're aiming.

3 WDL Examples

Listing 1 presents how the syntax expressed in the grammar can be used to specify tools and tasks. First, the tool's type signature is specified, then a set of execution attributes followed by how to execute the tool. A couple of conventions were adopted, for example, for a given resource named corpus.pt.en.tmx:

- $B1 is the name of the resource (corpus);
- $1 is the name of the resource coupled with its extension (corpus.pt.en.tmx).

It is also possible to specify variable filenames, or type extensions, as seen in the *tmx2tmxa* tool definition in listing 1. This way, the user can make variable parts of a filename part of the type definition (e.g. language pairs).

Table 1 Workflow Description Language Grammar.

wdl	→	'%tools' toolDefs '%tasks' taskSpecs
		;
toolDefs	→	toolName ':' signature tool
		| toolDefs toolName ':' signature tool
		;
signature	→	type_id '→' type_id
		;
tool	→	'{' action execAttrs split join '}'
		;
action	→	'@BASH' '{' STRING '}'
		| '@PERL' '{' STRING '}'
		;
execAttrs	→	execAttrs execAttr
		| ε
execAttr	→	'wall='wall
		| 'gpu=' bool
		| 'mic=' bool
		| 'ncores=' INT
		| 'nnodes=' INT
		| ...
		;
split	→	'split=' action
		| ε
		;
join	→	'join=' action
		| ε
		;
taskSpecs	→	taskSpecs task
		| ε
		;
task	→	toolName '(' args ')'
		| toolName '.' toolName '(' args ')'
		| '#make' '(' args ')'
		| '#span' '(' args ')'
		;
args	→	ID
		| args ',' ID
		;

Listing 1 WDL Example.

```
1
2  tmx2ptd : tmx -> ptd {
3      @BASH {
4          nat-create -id $B1.ptd -tmx $1
5      }
6  }
7
8  tmx2tmxa : $l1.$l2.tmx -> tmxa {
9      @BASH {
10         tmx2tmxa -l $l1-$l2 -f $1 -o $B1.$l1.$l2.tmxa
11     }
12 }
13
14 tmx2ptda : tmxa -> ptda {
15     @BASH {
16         tmxa-lemmatizer -i $1 -o $B1.tmxpl
17         nat-create -id $B1.ptda -tmx $B1.tmxpl
18     }
19 }
20
21 tmx2tmxa (corpus.pt.en.tmx)
```

Figure 2 Typical serial workflow.

An alternative to specifying tasks in a function call fashion is to write something like the following using function composition:

```
1  tmxa2ptda.tmx2tmxa (corpus.pt.en.tmx)
```

The resulting workflow can be seen in Figure 2.

For example, the following definition adds mapReduce primitives to the tmxa2ptda tool[1].

The resulting workflow can be seen in Figure 3.

More complex examples, given there are no dependencies, would allow tasks to execute in parallel in a Cluster environment, spawning several processes in a single machine, or even across multiple machines. This would allow for a more efficient use of the available computational resources. Communication is done preferably by rsync, since it is a safer and faster alternative to scp.

There is also ongoing work in modes in which the user writes:

```
1  #span(resource.src_type)
2  #make(resource.target_type)
```

The *#span* construct will traverse the graph generated from the type signatures of the tools and generate all the resources possible from the achievable nodes. The *#make* construct

[1] *tmxsplit* is a script that cuts a tmx when possible after a given number of lines. *ptdcat* is a script that concatenates the various parts of a ptd file.

Listing 2 WDL with MapReduce Example.

```
tmxa2ptda : tmxa -> ptda {
    @BASH {
        tmxa-lemmatizer -i $1 -o $B1.tmxpl
        nat-create -id $B1.ptda -tmx $B1.tmxpl
        }
        split=@BASH {
            tmxsplit -l 10000 $1
        }
        join=@BASH {
            ptdcat *.ptda > result.ptda
        }
}
```

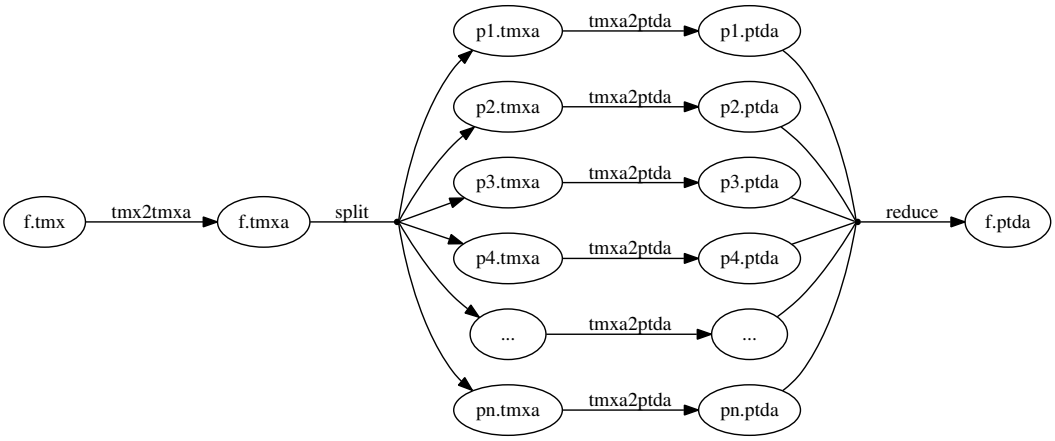

Figure 3 Workflow with MapReduce.

will take a *resource.target_type*, and, by inference, figure out the correct path to generate it from an existing *resource.src_type*. There will also be a built-in Perl preprocessor, so that the user can build more complex tasks.

4 Conclusions

Workflow orchestration and management is something very present in today's computing world and specially in fields that deal with intensive tools and very large data-sets. The lack of a tool to achieve this end, makes it even more difficult. In this short paper we introduced a new language to orchestrate workflows and to better manage existing computational resources. High level primitives and the direct integration of a scripting language (Perl) means we achieved a versatile format to express complex workflows.

Acknowledgments. This work is funded by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project PEst-OE/EEI/UI0752/2014

References

1. José João Almeida, Sílvia Araújo, Nuno Carvalho, Idalete Dias, Ana Oliveira, André Santos, and Alberto Simões. The Per-Fide corpus: A new resource for corpus-based terminology, contrastive linguistics and translation studies. In Tony Berber Sardinha and Telma São-Bento Ferreira, editors, *Working with Portuguese Corpora*, chapter 9, pages 177–200. Bloomsbury Publishing, April 2014.
2. M. Bhandarkar. MapReduce programming with Apache Hadoop. *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010.
3. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):1–13, 2008.
4. Mikel L. Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M. Tyers. Apertium: a free/open-source platform for rule-based machine translation. *Machine Translation*, 25(2):127–144, 2011.
5. David Jackson, Quinn Snell, and Mark Clement. Core Algorithms of the Maui Scheduler. In *Job Scheduling Strategies for Parallel Processing*, pages 87–102. Springer-Verlag, 2001.
6. Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL'07, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
7. K. H. Lee, Y. J. Lee, H. Choi, Y. D. Chung, and B. Moon. Parallel data processing with MapReduce: a survey. *ACM SIGMOD Record*, 40(4):11–20, 2012.
8. Lluís Padró and Evgeny Stanilovsky. FreeLing 3.0: Towards Wider Multilinguality. *LREC*, pages 2473–2479, 2012.
9. Alberto Simões, Rúben Fonseca, and José João Almeida. Makefile::Parallel Dependency Specification Language. In Anne-Marie Kermarrec, Luc Bougé, and Thierry Priol, editors, *Euro-Par 2007*, pages 33–41, Rennes, France, August 2007. Springer-Verlag.
10. Alberto M. Simões and J. J. Almeida. NATools – a statistical word aligner workbench. *Procesamiento del Lenguaje Natural*, 31:217–224, Sep. 2003.