SAASFEE: Scalable Scientific Workflow Execution Engine

Marc Bux*
Humboldt-Universität zu Berlin, Germany
bux@informatik.hu-berlin.de

Jörgen Brandt*
Humboldt-Universität zu Berlin, Germany
brandjoe@informatik.hu-berlin.de

Carsten Lipka
Humboldt-Universität zu Berlin, Germany
lipka@informatik.hu-berlin.de

Kamal Hakimzadeh
KTH Royal Institute of Technology, Sweden
mahh@kth.se

Jim Dowling
KTH Royal Institute of Technology, Sweden
jdowling@kth.se

Ulf Leser
Humboldt-Universität zu Berlin, Germany
leser@informatik.hu-berlin.de

ABSTRACT

Across many fields of science, primary data sets like sensor read-outs, time series, and genomic sequences are analyzed by complex chains of specialized tools and scripts exchanging intermediate results in domain-specific file formats. Scientific workflow management systems (SWfMSs) support the development and execution of these tool chains by providing workflow specification languages, graphical editors, fault-tolerant execution engines, etc. However, many SWfMSs are not prepared to handle large data sets because of inadequate support for distributed computing. On the other hand, most SWfMSs that do support distributed computing only allow static task execution orders. We present SAASFEE, a SWfMS which runs arbitrarily complex workflows on Hadoop YARN. Workflows are specified in Cuneiform, a functional workflow language focusing on parallelization and easy integration of existing software. Cuneiform workflows are executed on Hi-WAY, a higher-level scheduler for running workflows on YARN. Distinct features of SAASFEE are the ability to execute iterative workflows, an adaptive task scheduler, re-executable provenance traces, and compatibility to selected other workflow systems. In the demonstration, we present all components of SAASFEE using real-life workflows from the field of genomics.

1. INTRODUCTION

Over the last years, research in essentially all fields of science has become more and more data-intensive. The predominant way of analyzing scientific data is to use complex pipelines composed of highly specialized, domain-dependent tools. These tools are developed by a community of researchers in a multitude of different languages. General-purpose scripting languages (Python, Perl, Bash) are as common as domain-specific tools (R, MATLAB, LINPACK) and low-level programming languages for time-critical tasks (C, Fortran). SWfMSs have emerged to facilitate the design, implementation, execution, optimization, monitoring, and exchange of such heterogeneous pipelines.

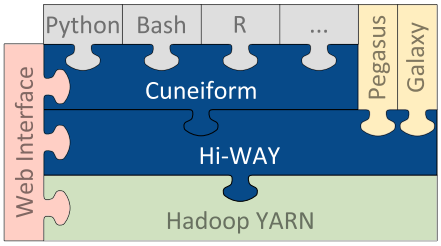

Figure 1: The SAASFEE software stack, comprising the Cuneiform language and Hi-WAY scheduler.

Existing SWfMSs can be roughly classified into two different groups [4]. Systems of the first group (e.g., Taverna, Kepler, Galaxy) focus on usability and ease-of-assembly, yet provide only (if any) limited horizontal scalability. Members of the second group emphasize distributed execution but typically lack the ability to run iterative workflows with control structures (e.g., Askalon, Pegasus). In addition, all current SWfMSs we are aware of come with their own proprietary execution engines which are not designed to manage all compute resources within a cluster and have been superseded by more general resource managers and schedulers, such as YARN [12] or MESOS [10]. Recently, a third class of systems, which in principle are suitable for executing scientific workflows, emerged around the MapReduce programming model (e.g., Spark [13], Stratosphere/Flink [1]). While these systems excel at providing abstractions for programming compute-intensive tasks, reusing and controlling software with no Java API is quite cumbersome. In summary, there exist no platforms that (a) can easily embrace the ever-evolving research tools developed and maintained by thousands of scientists, (b) scale to very large data sets, and (c) are able to execute arbitrarily complex workflows.

*These authors contributed equally to this work.

Proceedings of the VLDB Endowment, Vol. 8, No. 12

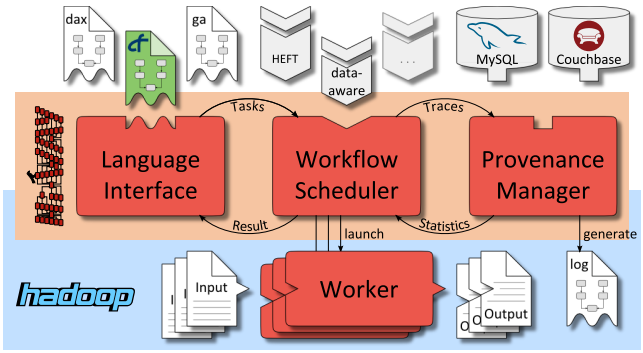

Figure 2: Hi-WAY architecture: the language interface parses a workflow and reports tasks to the scheduler, which is provided with statistics on previous task executions by the provenance manager.

Here, we demonstrate SAASFEE[1], a ScAlAble ScientiFic workflow Execution Engine (see Figure 1), which aims at closing this gap. SAASFEE workflows are specified in the functional workflow language Cuneiform [3]. Cuneiform allows for loops and conditionals (to support complex, iterative workflows) and can seamlessly integrate third-party tools and languages. Cuneiform scripts are executed by Hi-WAY, a higher-level scheduler for YARN, Hadoop's resource management component, from which Hi-WAY inherits its scalability and fault tolerance. Hi-WAY supports iterative workflows, adaptive scheduling to cope with virtualized execution environments, and re-executable provenance traces. Furthermore, Hi-WAY can execute workflows specified in languages other than Cuneiform, which reduces the need to run different platforms on the same cluster.

2. CUNEIFORM

Cuneiform is a functional language for specifying scientific data analysis pipelines at large. It focuses on algorithmic skeletons for deducing parallelization possibilities and easy reuse of existing software. To offer the highest amount of extensibility, Cuneiform has black-box data and operator models. Thus, integrating foreign code or passing around proprietary data formats comes at no cost. At the downside, a black-box operator model prevents optimization by task reordering, and a black-box data model requires developers to take care of format conversions. However, we find the former to be generally of little use in scientific workflows with their highly specialized operations (we focus on scheduling instead), and the latter imposing only moderate effort, since most workflows operate within a single domain (e.g., genomics, astrophysics) where usually a few different formats cover almost all use cases.

In Cuneiform, tasks are the basic unit of abstraction. The user can define and apply tasks like functions in functional programming languages. When a task is applied to a list of data items, the user determines whether this task is invoked separately for each element of the list, or only once, thereby consuming the list as a whole. Cuneiform also allows aggregation functions, making it straightforward to define MapReduce-like workflow structures. However, it goes well beyond MapReduce by allowing rich control structures within the language, which are also part of the scheduled execution plan. For instance, some applications necessitate a more general form of iteration than a *map*. Consider an iterative learning algorithm, which in each invocation consumes the output of the previous invocation until a convergence criterion is met. In Cuneiform, it is possible to express such an unbounded iteration by means of tail recursion with an exit condition. For instance, the following code snippet implements a k-means clustering in Cuneiform (presupposing low-level functions implemented as Python scrips):

```
deftask classify( labeled( File )
        : dataset( File ) meanset( File ) )in python
deftask refine( meanset1( File )
        : labeled( File ) )in python
deftask hasconverged( <converged> q1
        : dataset( File ) meanset( File ) q )in python
deftask kmeans( result( File )
        : dataset( File ) meanset( File ) q ) {
    labeled = classify( dataset: dataset
                        meanset: meanset );
    meanset1 = refine( labeled: labeled );
    converged q1 = hasconverged( dataset: dataset
                                 meanset: meanset1
                                 q:       q );
    result = if converged then meanset1
             else kmeans( dataset: dataset
                          meanset: meanset1
                          q:       q1 )
             end;
}
```

Cuneiform scripts are usually executed on distributed computational infrastructure using Hi-WAY (see next section). In addition, Cuneiform provides a local mode primarily intended for debugging and development purposes. However, this local mode fully exploits multicore machines and can automatically resume workflows after failure without repeating successfully finished tasks, making it a serious alternative for less compute-intensive runs. Cuneiform also provides an interactive console and an editor, which automatically generates the graphical representations of a workflow during specification (see Figure 4).

3. Hi-WAY

The prominent Apache Hadoop framework has recently been extended to support arbitrary programming models beyond MapReduce through its resource management component YARN. Hi-WAY is a higher-level scheduler for YARN specialized in executing scientific workflows. Hi-WAY executes each workflow task in its own container, which is YARN's basic unit of computation, encapsulating resources such as memory and virtual CPU cores (see Figure 2). Hi-WAY is capable not only of running Cuneiform scripts, but can also execute workflows from Pegasus [8], as well as from Galaxy [9], a popular platform for workflows from the field of bioinformatics. The system comes with a number of further features important for supporting scientific analysis.

First, repeatability of experiments is of utmost importance in science [6]. Accordingly, Hi-WAY generates a comprehensive provenance trace that logs events at different levels of granularity (file, task invocation, workflow run). Traces are written to HDFS by default but can also be stored in a NoSQL database or an RDBMS for convenient long-term storage and post-analysis – e.g., to determine the lineage of a data product of a workflow run. Traces are directly re-executable and are therefore, together with the workflow input, a verifiable proof of the way an analysis result was derived.

[1] http://saasfee.informatik.hu-berlin.de/

Second, many complex types of analyses require workflows with control structures such as loops and conditions. In general, the execution plan of such a workflow cannot be determined upfront, but is data-dependent and only emerges during workflow execution. Unlike most established SWfMSs, Hi-WAY supports iterative workflows and dynamically expanding execution plans (see Section 2).

Third, as no assumptions whatsoever can be made on the black-box tasks comprising a workflow, reordering task execution across data dependencies while guaranteeing equivalence of results (as in classical query optimization) is impossible. Instead, Hi-WAY focuses on optimizing task scheduling. To this end, it provides a set of adaptive scheduling algorithms tailored to running workflows on virtualized hardware, which are typically subject to sudden and unforeseeable changes in resource performance and availability (like network or local I/O bandwidth) [5].

Note that Hi-WAY is completely agnostic to the nature of the tasks it executes, as well as their implementations and I/O formats. A prerequisite to this feature is that the necessary tools are pre-installed on the cluster. To this end, configuration routines are comfortably supported in the form of Chef cookbooks, which can be run using the VM provisioning toolkit Karamel[2].

In a preliminary evaluation of Hi-WAY's scalability, we specified the variant calling workflow described in Section 4.2 in Cuneiform and in Apache Tez[3]. Tez enables execution of complex dataflows on YARN and is therefore probably the system most similar to Hi-WAY. The major differences are Tez's lower-level architecture, resulting in dataflows having to be programmed in Java, and that it builds on a key-value data model – two design decisions that severely impact its usability for scientific workflows. We ran both systems on a Hadoop installation across 24 compute nodes with 24 GB main memory and two Intel Xeon E5-2620 processors with 12 logical cores each. Each of the up to 576 concurrently running containers was provided with its own virtual processor core and 1 GB of main memory. The results shown in Figure 3 indicate that Hi-WAY scales well even past 500 concurrent tasks and also outperforms Tez. However, the main advantage of SAASFEE compared to Tez was development time, which was days in Cuneiform yet more than two weeks in Tez. Note that in our cluster, scalability beyond 96 containers was severely hampered by network bandwidth (1 GBit switch).

4. DEMONSTRATION

The demonstration encompasses three simple, introductory workflows and two more complex real-life workflows from genome research. It showcases how workflows are designed, edited, debugged, executed, re-executed, and monitored. In particular, users are introduced to (a) the user interface, foreign tool support, and parallelization capabilities of Cuneiform, and (b) the Hadoop integration, scalability, provenance capabilities, and multiple workflow language support of Hi-WAY. Users can interact by developing their own workflows, interactively debugging scripts, easily setting up cluster environments, running workflows on different (virtual) cluster configurations, and inspecting and rerunning workflow traces.

[2] http://www.karamel.io/
[3] http://tez.apache.org/

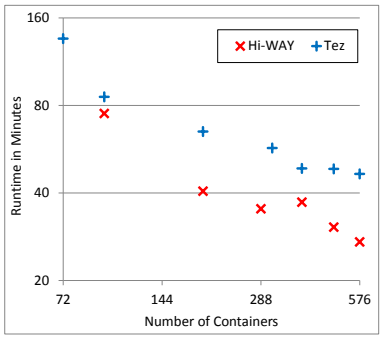

Figure 3: Mean runtimes of the variant calling workflow with increasing number of containers.

4.1 Simple workflows

We commence with a simple hello world example to introduce Cuneiform's language features regarding parallelization and foreign code integration. Since we are free in the choice of the scripting language in which we define a task, we can come up with, e.g., a *greet* task written in Bash or R:

```
deftask greet( out : person )in bash *{
    out="Hello $person"
}*
deftask greet( out : person )in r *{
    out = paste( "Hello", person )
}*
```

We next inspect a word count workflow similar to the well-known MapReduce example from [7]. We implement low-level functions for counting words in a text file (*map*) and for summing up the counts (*reduce*) in Python. In contrast to the hello world example, these functions now process files instead of strings. In the demonstration, we show how simple it is to produce variations of this workflow (filtering, dot and cross products, low-level tasks in other languages, etc.).

```
deftask count( counts( File ) : text( File ) )in python
deftask merge( sums( File ) : <counts( File )> )in python
```

Next, we analyze the Galaxy 101 workflow, a still comparably simple, yet real-life workflow computing distributions of mutations over genes. We show how straightforward it is to run this workflow (or other workflows designed in and exported from Galaxy) on Hi-WAY with its superior scalability. The only requirements are that input files are explicitly specified in the workflow and a Galaxy installation is available (but not necessarily running) on all worker nodes.

4.2 Complex workflows

As an example of a complex Cuneiform workflow, we explain and demonstrate a workflow for solving variant calling [11] (VC), an important step in the analysis of next-generation sequencing data. The problem is to determine and annotate variants in a genome, based on the output of a sequencing machine. The VC workflow encompasses index-based string similarity search, probabilistic reasoning to cope with the noisy data, and multiple database look-ups for finding related information. Scalability of this workflow is of major importance, as modern sequencing machines produce multiple terabytes of genomic data per week.

In the first step of the workflow, reads, the primary product of the sequencer, are aligned against a reference genome using the tool BWA. The resulting alignments are sorted using SAMtools, before variants are predicted with VarScan

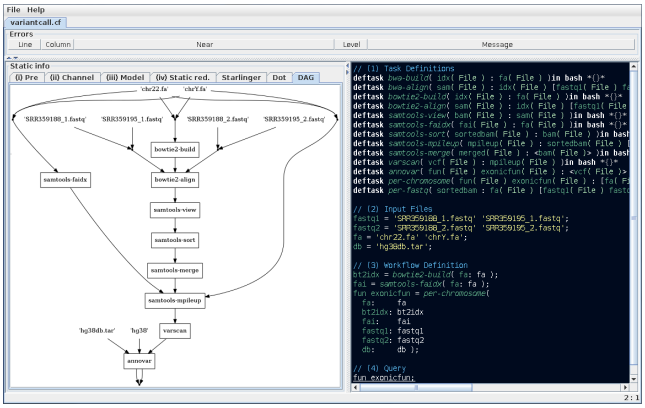

Figure 4: The VC workflow in the Cuneiform editor.

and finally annotated with ANNOVAR. Note that each of these steps can be (and, in practice, often is) performed by other similar tools. The workflow can be parallelized by partitioning the read set, the reference (e.g., into chromosomes), or both. In the demonstration, we use real read files from the 1000 Genomes project. Parts of the workflow and its high-level structure are displayed in Figure 4.

In the next step of this demonstration, we replace individual tools in the script to generate variations of this workflow, e.g., by using a read mapper other than BWA, aligning against another reference, increasing the amount of input (read) data, or using another data partitioning scheme. Note that determining optimal tool chains and parametrization for a given experimental setting is a highly relevant and complex topic in many fields of science, including genomics [2]; even slight modifications may have a dramatic impact and must be carefully tested. Providing such flexibility is thus very important for end users.

Our fifth and most complex demonstration workflow was also designed in Galaxy. The TRAPLINE RNAseq workflow[4] (see Figure 5) processes and compares RNA sequencing data samples. It consists of more than 60 tasks, has a degree of task parallelism of twelve, and takes hours to compute already for a single genomic sample due to very CPU-intensive tasks processing large amounts of data. Scaling out this workflow is important as current genomics projects easily produce hundreds of such samples. We show how this workflow can be exported from Galaxy and run on Hi-WAY, profiting from the provenance, fault tolerance, and horizontal scalability features of SAASFEE.

5. CONCLUSION

SAASFEE is a fully functional scientific workflow system which combines the flexibility of present SWfMSs with the scalability of Hadoop YARN. It offers a unique set of features tailored to making the scalable execution of scientific workflows on distributed hardware as simple as possible. SAASFEE is part of the BiobankCloud[5] stack, which also provides a web interface for running and monitoring workflows, easy provisioning and configuration of local or remote resources (e.g., Amazon's EC2), a workflow repository, a security toolset, and role-based access rights.

[4]http://www.sbi.uni-rostock.de/RNAseqTRAPLINE
[5]http://www.biobankcloud.com/

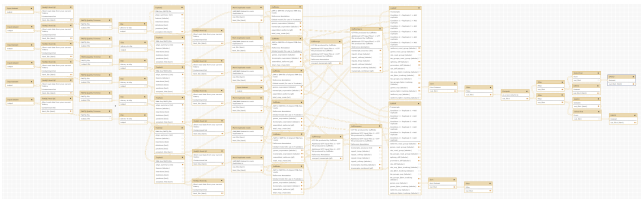

Figure 5: The TRAPLINE workflow in Galaxy's workflow repository visualized in its web interface.

6. FUNDING

M. Bux and J. Brandt are funded by the BiobankCloud EU project. We also acknowledge funding by the DFG graduate school SOAMED and an AWS in Education Grant.

7. REFERENCES

[1] A. Alexandrov et al. The stratosphere platform for big data analytics. *VLDB Journal*, 2014.

[2] T. S. Alioto et al. A comprehensive assessment of somatic mutation calling in cancer genomes. *bioRxiv*, 012997, 2014.

[3] J. Brandt, M. Bux, and U. Leser. Cuneiform: A functional language for large scale scientific data analysis. In *Proceedings of the Workshops of the EDBT/ICDT*, volume 1330, pages 17–26, 2015.

[4] M. Bux and U. Leser. Parallelization in scientific workflow management systems. *arXiv:1303.7195*, 2013.

[5] M. Bux and U. Leser. Dynamiccloudsim: Simulating heterogeneity in computational clouds. *Future Generation Computer Systems*, 2014.

[6] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD Conference*, pages 1345–1350, 2008.

[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] E. Deelman et al. Pegasus: A workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[9] J. Goecks, A. Nekrutenko, and J. Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[10] B. Hindman et al. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pages 295–308, 2011.

[11] S. Pabinger et al. A survey of tools for variant analysis of next-generation genome sequencing data. *Briefings in Bioinformatics*, 15(2):256–278, 2014.

[12] V. K. Vavilapalli et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the Fourth ACM Symposium on Cloud Computing*, 2013.

[13] M. Zaharia et al. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.