DAGwoman: enabling DAGMan-like workflows on non-Condor platforms

Heiko H. Schmidt[*]
Center for Integrative Bioinformatics Vienna (CIBIV)
Max F. Perutz Laboratories (MFPL),
University Vienna,
Medical University Vienna,
University for Veterinary Medicine Vienna,
Vienna; Austria
heiko.schmidt@univie.ac.at

Thomas Tschager
Center for Integrative Bioinformatics Vienna (CIBIV)
Max F. Perutz Laboratories (MFPL),
University Vienna,
Medical University Vienna,
University for Veterinary Medicine Vienna,
Vienna; Austria

ABSTRACT

Scientific analyses have grown more and more complex. Thus, scientific workflows gained much interest and importance to automate and handle complex analyses. Tools abound to ease generation, handling and enactment of scientific workflows on distributed compute resources. Among the different workflow engines DAGMan seems to be widely available and supported by a number of tools. Unfortunately, if Condor is not installed users lack the possibility to use DAGMan. A new workflow engine, DAGwoman, is presented which can be run in user-space and allows to run DAGMan-formatted workflows. Using an artificial and two bioinformatics workflows DAGwoman is compared to GridWay's GWDAG engine and to DAGMan based on Condor-G. Showing good results with respect to workflow engine delay and features richness DAGwoman offers a complementary tool to efficiently run DAGMan-workflows if Condor is not available.

Keywords

scientific workflows; workflow engine; DAGwoman; DAG-based workflows; bioinformatics

1. INTRODUCTION

In recent years computational analysis strategies have grown utterly complex incorporating various steps and approaches. Bioinformatics studies constantly have to deal with complex strategies as they may typically include steps like data collection or integration and quality control. They often employ a number of different analyses and finally the assessment of statistical significance of the results. As bioinformatics problems originate from various areas like genomics, proteomics, phylogenetics and phylogenomics, structure and functional prediction, handling huge next-generation sequencing data etc., also the analysis strategies are plentiful and highly diverse. Recently the field of scientific workflows [24] has gained a lot of momentum trying to cope with modeling and handling complex analysis workflows and their dependencies and to distribute them onto compute resources typically using schedulers or Grid middleware systems. Some aims of scientific workflow research are to ease automation of complex analyses as well as facilitating re-use of existing workflows. Scientific workflow enactment typically involves a number of tools at different levels. First, the workflow have to be modeled or implemented. When a workflow is executed, its inherent dependencies have to be controlled by a workflow engine. This engine submits tasks to a scheduling software or middleware (in grid systems) as soon as their dependencies are fulfilled. The scheduler then controls the execution of the tasks when free resources become available. For each layer software abound [27, 23, 5] optimizing different needs. There are graphical interfaces like Triana [15], Kepler [14] or Pegasus [4]. Some of these produce input to dedicated workflow engines like DAGMan [3, 4] from the Condor middleware [26] or use, e.g., Pegasus to do so [17]. The separate tasks are often distributed to the compute resources by scheduling systems like Condor [26] or Sun Grid Engine (SGE [11]). On distributed systems such as computational Grids Condor [26] and the Globus Toolkit [9] are used to connect distributed resources. While Condor and Globus Toolkit are commonly used on Grid systems, there are many clusters and institutional compute resources which solely rely on and offer schedulers like SGE. On such resources DAGMan is unfortunately not available to the users.

Here we present a DAG-based workflow manager, DAGwoman, that can run workflows specified in the DAGMan format on systems not running Condor or Condor-G. We will use a toy workflow and two bioinformatics workflows to measure the workflow engine delays of DAGwoman and other systems and discuss implications on their performance.

[*]corresponding author

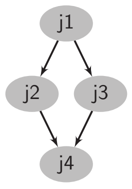
2. RELATED WORK

Since the late 1980ies the Condor Project has developed a framework to operate distributed computing and Grid resources [26, 3]. If administrators decide not to base on Condor, its job management module Condor-G can be used with simple.dag file:

```
JOB j1 script1
JOB j2 script2
JOB j3 script3
JOB j4 script4
PARENT j1 CHILD j2 j3
PARENT j2 CHILD j4

PARENT j3 CHILD j4

PRIORITY j2 100
CATEGORY j1 cat1
CATEGORY j2 cat1
MAXJOBS cat1 1
```



splice.dag file:

```
JOB start startscript
JOB end endscript
SPLICE A simple.dag
SPLICE B simple.dag
PARENT start CHILD A B
PARENT A B CHILD end

PRE start init.sh
POST end cleanup.sh
```

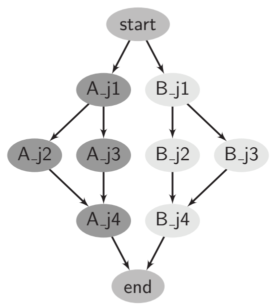


Figure 1: DAG-file examples.

middleware toolkits like Globus [10, 9]. A decade ago Condor introduced a Directed Acyclic Graph Manager (DAGMan [26, 3]) as a workflow engine for large workflows. DAGMan is used by several workflow softwares either directly [4, 16, 6] or indirectly though programs like Pegasus [17, 15]. DAGMan offers an easy-to-generate format to model scientific workflows as directed acyclic graphs (DAG) which is also supported by a number of other tools like Pegasus and Triana. Unfortunately, DAGMan is only available for systems running Condor/Condor-G [26, 10] which DAGMan is part of. DAGMan's DAG file format implements a variety of keywords to specify workflows. Some of the most important will be explained with Fig. 1. The nodes of the DAG structure are defined with the `JOB` keyword, while their parent-child dependencies are defined using `PARENT-CHILD`. Each of the nodes can have a pre-script and a post-script attached (cf. `PRE` and `POST` lines in Fig. 1). DAGMan offers keywords to assign priorities to jobs (`PRIORITY`). In addition one can assign each job to a category (`CATEGORY` line). Throttling can be applied to these categories by setting the maximal number of their jobs to be run simultaneously (`MAXJOBS`). Furthermore, DAGMan allows to easily build large workflows from pre-existing smaller ones by using the `SPLICE` in Fig. 1 the simple.dag workflow is sliced twice into the splice.dag workflow leading to the larger DAG on the right. Also does DAGMan implement an elaborate system to decide whether a job ran successfully or not based on the exit status of the job and its pre and post-script. In case that some nodes failed DAGMan does write a rescue DAG file where all nodes which could be run successfully are marked `DONE`. Rescue DAG files avoid the necessity of recomputing successful tasks if the workflow is re-run to be finished.

The ability of specify and manage complex workflows, is an important feature. Thus, the GridWay project [12] added a workflow engine GWDAG to handle DAGMan workflows, what they have described in the release notes of GridWay 5.3 as an almost de-facto standard to represent DAG workflows. GridWay is a metascheduler which like Condor-G uses the Globus Toolkit [9] to access various schedulers. This way DAGMan-based workflows are made accessible to a large variety of systems. From the functionality of DAG files, GWDAG implements the keywords `JOB` and `PARENT-CHILD`. Although the manual does not mention `PRE` and `POST` scripts, they seem to be recognized by GWDAG. Hence, GWDAG can recognize the basic workflow structure, but it seems to lack the elaborate DAGMan functionality of error handling as well as throttling the access to the scheduling system or to set priorities to certain tasks in the workflow.

3. WORKFLOW OVERHEAD

As typically several layers of software are involved in running a scientific workflow they all have an influence on the runtime of a workflow [19, 2]. The time between the release of a job, i.e. when all parental jobs it depends on are finished, and the submission of the job is called the *workflow engine delay* as it is mainly caused by the workflow engine itself. The *queue delay* between the job submission and the start of the job execution is usually determined by the performance of the scheduling system and mainly by the availability of resources. Since Grid middleware systems usually have their own transfer queues to keep tasks during match making, this also adds to the *queue delay*. It is followed by the actual *runtime* of the job and can be followed by a *postscript delay* if there is a postscript to clean up data or determine the exit state of the job. Naturally, queue delay, runtime and postscript delay are mainly caused by the external factors like load, speed and availability of the computing resources. Since we will mainly focus on the performance of the workflow engine DAGwoman which can run in user-space, we will concentrate on the workflow engine delay as it reflects the performance of the workflow engine itself. For comparison we will obtain workflow engine delays for DAGMan using Condor-G and GWDAG using GridWay and a GridWay-independent GWDAG (see below).

4. DAGWOMAN

DAGwoman allows to access a scheduling system either via command-line (CLI) or using the Distributed Resource Management Application API (DRMAA [20, 25]). It is written in Java and has been implemented in a modular way (Fig. 2) using plugins to encode different features. The Main class initializes the plugins implementing DAG interface and Submitter. Both classes inherit their content from plugins. DAG plugins encode the DAG data structure and implement the interface to query it. The submitter plugins implement the access to interfaces of scheduling systems.

The SGE commandline interface plugin uses the `qsub` command to submit to Sun/Oracle Grid Engine (SGE/OGE). To query the status of tasks `qacct` is called, because `qstat` can inquire for finished jobs, but does not show whether their final status was successful. The DRMAA plugin accesses the scheduler using the Distributed Resource Management Application API.

After the parser has read the DAG file, a DAG is initialized representing the workflow. Besides the characteristics of each task like pre and post scripts, priority, category, etc., each node knows its children and parents and for every node we use a dependency counter that is initialized with the number of dependencies. Whenever a node is finished, the dependency counters of all children are decremented. This is straightforward and allows to detect immediately whether

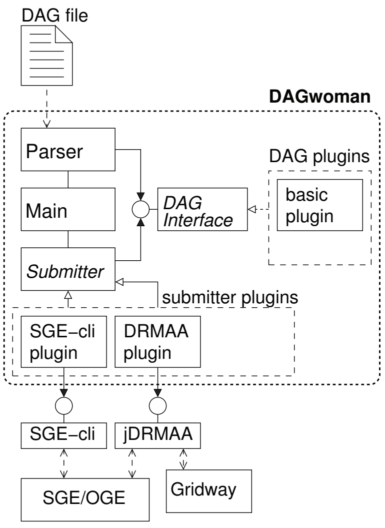


Figure 2: DAGwoman structure presenting its classes, plugin-structure and interfaces used.

a task is submittable. If a dependency counter gets zero, the respective task is submitted to a priority queue at once. After having reported a finished node the submitter inquires for submittable jobs. If there are any, they are extracted from the priority queue (respecting possible specified throttling) and submitted to the scheduling system. This way submittable jobs do not have to wait until all finished nodes have been processed, but are available for execution as early as possible.

The implemented parser class recognizes most of the original DAGMan-format features for job description with pre and post scripts, job categories, job and category priorities as well as DAG splicing. Furthermore, DAGwoman fully supports rescue DAGs which avoid re-running already successfully computed tasks in case of errors during workflow execution. Moreover, all parameters for job throttling are implemented allowing for the restriction of the sum of all jobs or the amount of jobs of a certain category allowed to be submitted to the scheduling system. It also allows to restrict the number of simultaneous pre and post scripts to avoid overloading the machine running the workflow engine. The user is free to specify any prioritization scheme, however, DAGwoman implements a few simple prioritization mechanisms like using a task's longest path to a DAG's leaf or the sum of tasks depending on the current node in the subsequent part of the workflow.

While DAGwoman has mainly been used and tested in a Sun Grid Engine setup, its modular implementation and DRMAA allows to extend the number of schedulers accessible.

5. WORKFLOWS AND SETUP

5.1 The Workflows

We use three different workflows to check the performance of the DAGwoman implementation, namely one toy workflow and two real-world bioinformatics workflows from the field of phylogenomics.

The Delay Workflow. In bioinformatics analyses often large sets on independent tasks are run and later evaluated. This

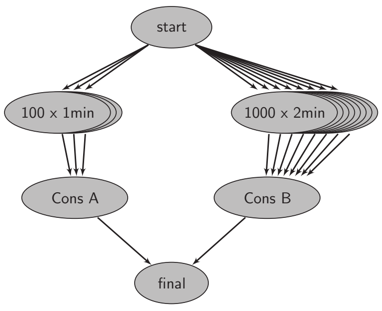


Figure 3: The Delay Workflow has two sub-DAGs with 100 tasks of 1 min and 1000 tasks of 2 min.

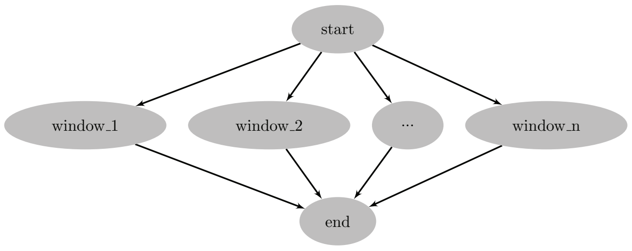


Figure 4: The RecDetec Workflow performs a sliding window strategy to determine possible recombination break points in viral genomes.

is best reflected by a diamond-shaped DAGs with many nodes. We have seen earlier that often the sheer number of active jobs delays the determination of submittability in another analyses. To emulate this scenario use a simple workflow with two diamond shaped sub-DAGs (Fig. 3), one contains 100 tasks of 1 min each and another comprising 1000 tasks of 2 min. We want to figure out whether the determination that all dependencies of task *Cons A* are fulfilled is delayed.

The RecDetec Workflow. This workflow performs a sliding window strategy (Fig. 4), named bootscanning [21] to determine possible recombination break points [22] in viral genomes. This is an analysis routinely performed in virology research and epidemiology especially when dealing with RNA viruses [22]. In the test setup an alignment of Rubella virus genomes was sliced into about 400 overlapping windows. For each window 100 maximum likelihood bootstrap tree reconstructions were performed [18, 8].

The PhyloWorkflow. The final workflow (Fig. 5) aims at reconstructing the evolutionary relationships of a set species using sequences of several genes employing a variety of phylogenetic methods. The PhyloWorkflow takes as input a set of gene alignments and performs a large variety of phylogenetic reconstructions based of 48 different variants of supertree, superalignment and medium-level methods [1, 7, 13]. Some of the methods require at least 100 bootstrap analyses which are stacked nodes marked with B in the graph. Since the analysis uses several gene alignments as input the number of stacked nodes marked with G depends on the number of genes. In the test setup this led to a full workflow of 565 single tasks to be performed.

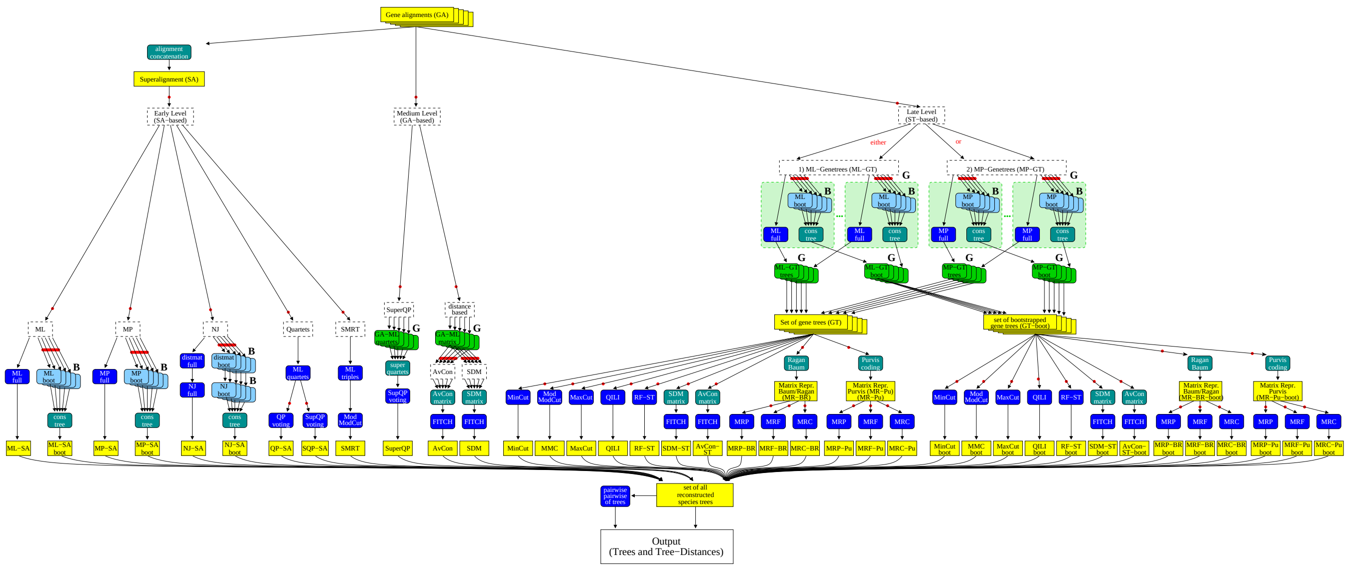


Figure 5: The PhyloWorkflow performes a phylogenomic analysis to reconstruct species trees.

5.2 Compute Setup

The workflows were run using DAGwoman submitting either via DRMAA or through the command-line interface to the underlying scheduling system.

The compute resource contains (a) 34 nodes with a total of 136 cores with 2GB RAM per core, (b) 14 nodes with a total 112 cores with 6GB per core, and (c) 4 nodes with a total of 96 cores with 128GB per node. Thus, there were 344 cores available which were managed through a Sun Grid Engine [11]. The tests were performed during normal daytime load. The workflow engine delays were obtained from the softwares' log files. For the middleware-based workflow engines Globus only had a single resource available to submit jobs to. For comparison Condor-G was installed using Globus Toolkit and we used DAGMan to execute the workflows. The number of jobs per submission cycle have been increased to a 5-fold of the default values. We also used GWDAG with GridWay and Globus Toolkit. Also for GridWay the number of jobs per submission cycle have been increased to a 5-fold of the default values. GWDAG is implemented in Ruby and interacts with GridWay using the commandline interface. In addition, we used a GWDAG software where the Ruby code was changed to use the SGE commandline interface to have another workflow engine which is able to run in user space. Submission and inquiring job success was done using `qsub` and `qacct`. Whether a job had finished was determined with `qstat -s z`.

In the following we refer to the workflow engines as DAGwoman-DRMAA, DAGwoman-SGE, DAGMan, GWDAG-GW and GWDAG-SGE, respectively. In some figure the lables had to be abbreviated in an unambiguous way.

6. RESULTS

The results of the Delay Workflow (Fig. 6), the RecDetec Workflow (Fig. 7) and the PhyloWorkflow (Fig. 8) show the workflow engine delays. In the bottom part the delays are sorted in the same order the tasks were submitted by the workflow engines. The top parts summarize the delays in

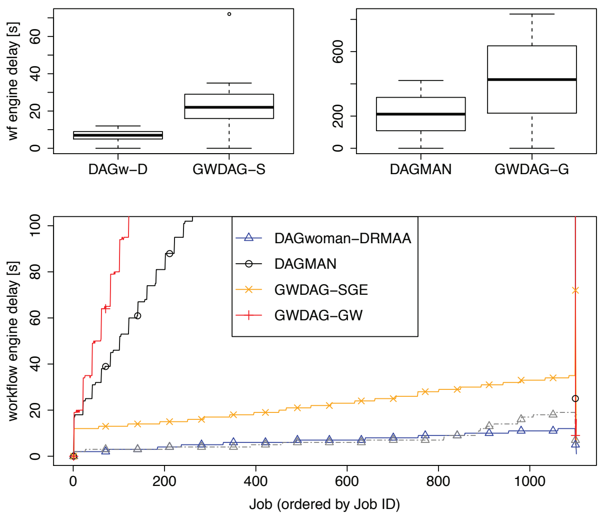


Figure 6: Results of the Delay Workflow (Fig. 3).

boxplots, because the lower plots had to be limited to a maximum of 150s to be able to distinguish the smaller delays.

The results of all three workflows show that DAGwoman-DRMAA out-performs the other engines with respect to the workflow engine delay. Due to the throttling implied by default by the two middleware-based workflow engines their delays exeeded the others often by an order of magnitude. Note, that the additional times for match making are not included in these values.

The Delay Workflow (Fig. 6) shows a delay time which is about a factor of 2 faster for DAGwoman (using DRMAA) compared to the GridWay-independent GWDAG-SGE. GWDAG-SGE exhibits a high peak for a single job with a delay between the fulfillment of the dependencies and the submission of about 1 min. The peak denotes the *Cons*

A job. From the length of the delay and the submission at the end of the workflow one tends to speculate, that this may be caused by an ineffective status inquiry. According to the log files *Cons A* started after all other jobs had almost finished. If this was a long job it would add to the running time of the workflow. The middleware-based DAGMan and GWDAG-GW are hampered by the default throttling of the number of jobs per submission cycle, although this had been increased by 5-fold compared to the default.

The grey line rising over the DAGwoman-DRMAA in (Fig. 6) was one specific run where after about job 900 the delay time suddenly rose. Close investigation showed that this was caused by a mass submission of jobs via a command-line script by a user of the compute resource. This reduced the responsiveness of the scheduling system. The other lines were computed from average workflow runs which were not hampered by such an event. The event however clearly shows that the workflow engine delay may not only be caused by the workflow engine. The delay time may also suffer if the responsiveness of the scheduler is heavily reduced. However, we only observed such behavior once.

In the RecDetec Workflow (Fig. 7) one can see a similar behavior showing that the workflow engine delays again grow with the number of tasks in the system, but most pronounced again for DAGMan and GWDAG-GW due to their default throttling. Comparing DAGwoman-DRMAA and the command-line version (SGE, in Fig. 7) shows that submission via the DRMAA API generally shows a better performance compared to submission via the command-line. This can partly be attributed due to the use of the SGE accounting system through `qacct`, because it obtains the information only after the task has left the queue and thus later than DRMAA which interacts with the SGE master directly. Note, that GWDAG-SGE submits and queries the jobs also via the command-line, however, its delay times exceeds that of DAGwoman-SGE usually by a couple of seconds.

The PhyloWorkflow (Fig. 8) shows a much more complex scenario. While the DAGwoman-DRMAA version always

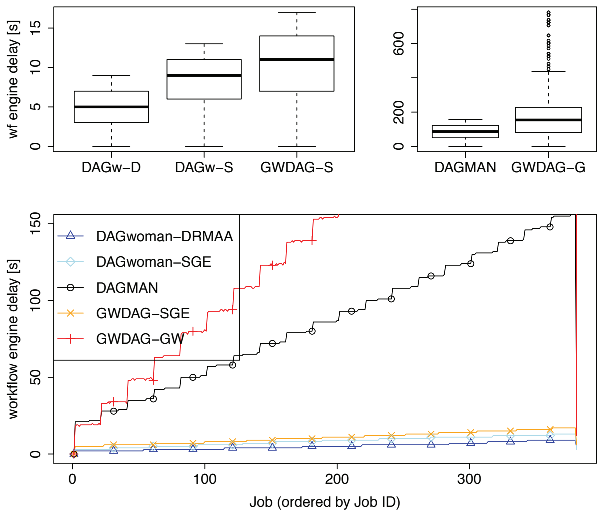


Figure 7: Results of the RecDetec Workflow (Fig. 4).

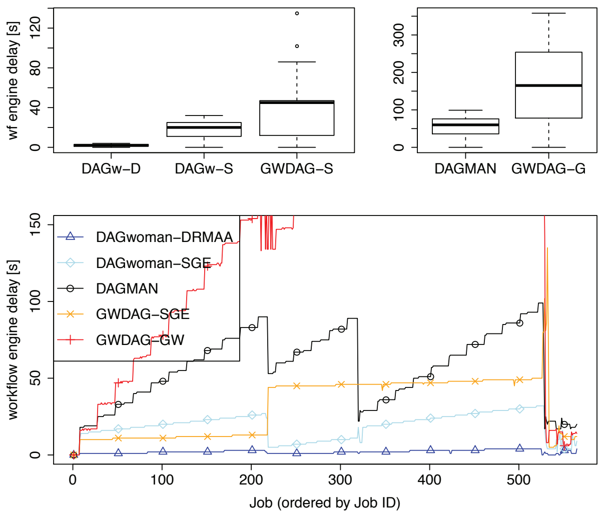


Figure 8: Results of the PhyloWorkflow (Fig. 5).

shows the shortest workflow engine delay, GWDAG-SGE is better than DAGwoman-SGE for more than the first 220 tasks, then GWDAG-SGE's delay rises to about 43s while those of DAGwoman-SGE drops considerably before rising again. Why this happens we could not figure out, but 300 bootstrap analyses are started at task 220 while 200 bootstrap analyses are already in the queue. Interestingly DAGMan show much better delay times than GWDAG-GW.

7. CONCLUSIONS

The results have shown that DAGwoman-DRMAA always shows the best workflow engine delay while DAGwoman-SGE and GWDAG-SGE show comparable results. The workflow engine delays of DAGMan and GWDAG-GW are typically one order of magnitude higher than those of the other three. This comparison, however, is unfair, because the higher delay is caused by settings which are necessary and tuned for large Grid systems. One has to keep in mind that DAGwoman does not aim at substituting any of the two in Grid environments. DAGwoman focuses at smaller compute resources enabling users to run DAG workflows which is not available otherwise if Condor, Condor-G or GridWay are not installed. If Condor/Condor-G is installed and optimized for the local system, typically DAGMan is the best choice.

In the case that one wants to perform DAG file based workflows DAGwoman should be preferred over the GridWay-independent version GWDAG-SGE. While both can handle workflows in DAGMan format, DAGwoman presented the better performance and the more extensive functionality. DAGwoman has shown promising results, enabling users to run DAGMan workflows efficiently on systems where it was not possible before unless the administrators agreed to install Condor or Condor-G in addition to their preferred scheduling system.

8. ACKNOWLEDGMENTS

We thank Stefan Ganscha, Christian Margreiter, Moritz Reinhard, David Szkiba for implementing the PhyloWorkflow together with TT and also Moritz Reinhardt for implementing the RecDetec Workflow. We thank Wolfgang Fischl for mak-

ing GWDAG, GridWay, Condor-G and Globus work. DAG-woman is available from `http://www.cibiv.at/software`.

9. REFERENCES

[1] O. R. P. Bininda-Emonds, editor. *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*. Kluwer Academic, Dordrecht, 2004.

[2] W. Chen and E. Deelman. Workflow overhead analysis and optimizations. In *Proceedings of the 6th workshop on Workflows in support of large-scale science (WORKS 2011)*, pages 11–20, New York, 2011. ACM.

[3] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger. Workflow in Condor. In I. J. Taylor, E. Deelman, D. Gannon, and M. S. Shields, editors, *Workflows for e-Science*, pages 357–375. Springer, London, 2007.

[4] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *Proceedings of the 2nd European AcrossGrids Conference (AxGrids 2004)*, volume 3165 of *Lecture Notes in Computer Science*, pages 131–140, Berlin, 2004. Springer.

[5] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Futur. Gener. Comp. Syst.*, 25:528–540, 2009.

[6] E. Deelman, M. Livny, G. Mehta, A. Pavlo, G. Singh, M.-H. Su, K. Vahi, and R. K. Wenger. Pegasus and DAGMan from concept to execution: Mapping scientific workflows onto today's cyberinfrastructure. In L. Grandinetti, editor, *High Performance Computing and Grids in Action*, volume 16 of *Advances in Parallel Computing*, pages 56–74. IOS Press, Amsterdam, 2008.

[7] I. Ebersberger, A. von Haeseler, and H. A. Schmidt. Phylogenetic reconstruction. In T. Lengauer, editor, *Bioinformatics – From Genomes to Therapies*, volume 1, pages 83–128. Wiley-VCH Verlag, Weinheim, Germany, 2 edition, 2006.

[8] J. Felsenstein. Confidence limits on phylogenies: An approach using the bootstrap. *Evolution*, 39:783–791, 1985.

[9] I. Foster. Globus toolkit version 4: Software for service-oriented systems. *J. Comput. Sci. Technol.*, 21:513–520, 2006.

[10] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Comput.*, 5:237–246, 2002.

[11] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID 2001)*, page 35, Washington, DC, USA, 2007. IEEE Computer Society.

[12] J. Herrera, E. Huedo, R. S. Montero, and I. M. Llorente. Porting of scientific applications to grid computing on gridway. *Sci. Prog.*, 13:317–331, 2005.

[13] A. Kupczok, H. A. Schmidt, and A. von Haeseler. Accuracy of phylogeny reconstruction methods combining overlapping gene data sets. *Algorithms Mol. Biol.*, 5:37, 2010.

[14] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, and Y. Jing, Taoand Zhao. Scientific workflow management and the kepler system). *Concurr. Comput.-Pract. Exp.*, 18:1039–106, 2005.

[15] S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang. Triana: A graphical web service composition and execution toolkit. In *Proceedings of the IEEE International Conference on Web Services (ICWS 2004)*, pages 514–524, Los Alamitos, CA, 2004. IEEE Computer Society.

[16] G. Malewicz, I. Foster, A. L. Rosenberg, and M. Wilde. A tool for prioritizing DAGMan jobs and its evaluation. *J. Grid Comput.*, 5:197–212, 2007.

[17] N. Mandal, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Integrating existing scientific workflow systems: The kepler/pegasus example. In *Proceedings of the 2nd Workshop on Workflows in Support of Large-Scale Science (WORKS'07)*, pages 21–28, New York, 2007. ACM.

[18] B. Q. Minh, L. S. Vinh, A. von Haeseler, and H. A. Schmidt. pIQPNNI – parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, 21:3794–3796, 2005.

[19] R. Prodan and T. Fahringer. Overhead analysis of scientific workflows in Grid environments. *IEEE Trans. Parallel Distrib. Syst.*, 19:378–393, 2008.

[20] H. Rajic, R. Brobst, W. Chan, F. Ferstl, J. Gardiner, A. Haas, B. Nitzberg, D. Templeton, P. Tröger, J. Tollefsrud, and A. Haas. Distributed resource management application api specification 1.0 (gwd-r.133). online www.drmaa.org, June 2008.

[21] M. O. Salminen, J. K. Carr, D. S. Burke, and F. E. McCutchan. Identification of breakpoints in intergenotypic recombinants of HIV type 1 by bootscanning. *AIDS Res. Hum. Retroviruses*, 11:1423–1425, 1995.

[22] E. Simon-Loriere and E. C. Holmes. Why do RNA viruses recombine? *Nat. Rev. Microbiol.*, 9:617–626, 2011.

[23] C. Stratan, A. Iosup, and D. H. J. Epema. A performance study of grid workflow engines. In *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing, (GRID 2008)*, pages 25–32, Washington, DC, USA, September 2008. IEEE Computer Society.

[24] I. J. Taylor, E. Deelman, D. Gannon, and M. S. Shields, editors. *Workflows for e-Science*. Springer, London, 2006.

[25] D. Templeton, P. Tröger, R. Brobst, A. Haas, H. Rajic, and J. Tollefsrud. Distributed resource management application API JavaTM language bindings 1.0. online www.drmaa.org, January 2007.

[26] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurr. Comput.-Pract. Exp.*, 17:323–356, 2005.

[27] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *J. Grid Comput.*, 3:171–200, 2006.