2014

A Comparison of a Campus Cluster and Open Science Grid Platforms for Protein- Guided Assembly using Pegasus Workflow Management System

Natasha Pavlovikj
University of Nebraska-Lincoln, npavlovikj@cse.unl.edu

Kevin Begcy
University of Nebraska-Lincoln, kevinbegcy@gmail.com

Sairam Behera
University of Nebraska-Lincoln, sbehera@cse.unl.edu

Malachy Campbell
University of Nebraska-Lincoln, campbell.malachy@gmail.com

Harkamal Walia
University of Nebraska-Lincoln, hwalia2@unl.edu

See next page for additional authors

2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops
Pages: 546 - 555, DOI: 10.1109/IPDPSW.2014.66
IEEE Conference Publications

A Comparison of a Campus Cluster and Open Science Grid Platforms for Protein-Guided Assembly using Pegasus Workflow Management System

Natasha Pavlovikj[1], Kevin Begcy[2], Sairam Behera[1], Malachy Campbell[2], Harkamal Walia[2], Jitender S. Deogun[1]

[1]Department of Computer Science and Engineering,
University of Nebraska-Lincoln
Lincoln, NE 66588-0115
Email: {npavlovikj, sbehera, deogun}@cse.unl.edu

[2]Department of Agronomy and Horticulture,
University of Nebraska-Lincoln
Lincoln, NE 68583
Email: {kevinbegcy, campbell.malachy, harkamal.walia}@gmail.com

***Abstract*—Scientific workflows are a useful tool for managing large and complex computational tasks. Due to its intensive resource requirements, the scientific workflows are often executed on distributed platforms, including campus clusters, grids and clouds. In this paper we build a scientific workflow for *blast2cap3*, the protein-guided assembly, using the Pegasus Workflow Management System (Pegasus WMS). The modularity of *blast2cap3* allows us to decompose the existing serial approach on multiple tasks, some of which can be run in parallel. Afterwards, this workflow is deployed on two distributed execution platforms: Sandhills, the University of Nebraska Campus Cluster, and the Open Science Grid (OSG). We compare and evaluate the performance of the built workflow for the both platforms. Furthermore, we also investigate the influence of the number of clusters of transcripts in the *blast2cap3* workflow over the total running time. The performed experiments show that the Pegasus WMS implementation of *blast2cap3* significantly reduces the running time compared to the current serial implementation of *blast2cap3* for more than 95 %. Although OSG provides more computational resources than Sandhills, our workflow experimental runs have better running time on Sandhills. Moreover, the selection of 300 clusters of transcripts gives the optimum performance with the resources allocated from Sandhills.**

Keywords—scientific workflow; pegasus workflow management system; transcriptome assembly; protein-guided assembly; blast2cap3; campus cluster; open science grid

I. INTRODUCTION

The advances in life sciences and information technologies have led to proliferation of scientific data that needs to be stored and analyzed. The analysis of this so called "big data" is done by using a complex set of multitude of software tools. A sequential series of these tools is known as an *analysis pipeline* [29]. The "big data" is too large to be processed by using only local computational resources. A possible approach to this problem is to make better use of multiple distributed resources including multi-core computers.

Scientists use various workflow systems to conduct their research modularly. This indicates that the whole scientific workflow can be decomposed into multiple sub-workflows that can be executed in parallel on distributed resources. Each workflow is composed of computational tasks, the order of execution of which is determined by the dependencies among the tasks [1]. The advantages of scientific workflows include automated complex analysis, real-time results and improved time performance that allow scientists to easily design, execute, debug, modify and re-run their experiments [17].

Over the past decade, several scientific workflows have been created and introduced. Pegasus Workflow Management System (Pegasus WMS) automatically maps high-level scientific workflows organized as directed acyclic graph (DAG) onto available distributed resources [2]. DAGMan (Directed Acyclic Graph Manager) is a meta-scheduler that submits jobs to Condor [4] in an order defined in DAG, and processes the results afterwards [3]. Taverna [5] is an open source workflow system that graphically connects bioinformatics web services together into a coherent flow. Kepler [6] also has a visual interface and separates the structure of the workflow model from its model of computation. The number of applications using scientific workflow systems has been steadily increasing [7].

The resources required by scientific workflows may exceed the capabilities of the local computational resources. Therefore, the scientific workflows are usually executed on distributed platforms, such as campus clusters or grids. Grids such as Open Science Grid (OSG) [8] and XSEDE [9] allow distributed computing where the computational resources are spread on a geographically remote location. Beside the cluster and grid execution platforms, lately the scientists are analyzing the benefits of using clouds for these scientific workflows. Cloud computing platforms like the commercial Amazon Elastic Compute Cloud [10] or the academic

978-1-4799-4116-2/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPSW.2014.66

FutureGrid [11] provide rentable computational and storage resources over the Internet. Despite the advantages and disadvantages of clusters, grids and clouds [30], the execution of scientific workflows deals with different challenges depending on the chosen computational platform.

In this paper we build a scientific workflow for *blast2cap3*, the protein-guided assembly, using Pegasus WMS. We chose two execution platforms for this workflow that represent the campus cluster, Sandhills, and the Open Science Grid. Furthermore, we compare the running time and used resources for the both platforms when the workflow is executed serially and parallel with alternating number of tasks.

This paper is organized as follows. In Section 2 we describe *blast2cap3,* the protein-guided assembly. Pegasus Workflow Management System is described briefly in Section 3. Section 4 includes overview of the used execution platforms, the campus cluster and the Open Science Grid. The implementation of the experiments used in this paper is presented in Section 5. In Section 6 we evaluate the built workflow and in Section 7 we draw conclusions based on our results and performed evaluation.

II. *BLAST2CAP3*: PROTEIN-GUIDED ASSEMBLY

Gene expression and transcriptome analysis are currently one of the main focuses of research for a great number of biologists and scientists. However, the assembly of raw sequence data to obtain a draft transcriptome of an organism is a complex multi-stage process usually composed of preprocessing, assembling, and post-processing. Each of these stages includes multiple steps such as data cleaning, contaminant removal, error correction, de novo assembly, redundancy reduction, and assembly validation. An *assembly pipeline* is used to simplify the entire assembly process by automating the most steps of the pipeline for producing correct transcripts [15]. A general transcriptome assembly pipeline with some common steps and the tools used for those steps is shown on Fig. 1.

After the data is cleaned and filtered in the preprocessing stage, the next step is to generate transcriptome assembly from the filtered reads. Multiple approaches used for assembling the filtered reads [12] produce high redundancy of the resulting transcripts. Therefore, these transcripts need to be merged into larger ones in order to remove redundancy. Overlap-based assembly program CAP3 is used to merge transcripts based on the overlapping region with specified identity [13]. However, the number of transcripts that need to be merged sometimes overwhelms the memory and time limits of CAP3. Additionally, CAP3 merges transcripts based on only nucleotide similarity that can lead to incorrect results because most of the generated transcripts code for a protein. Hence, protein similarity should be considered when the transcripts are merged.

Blast2cap3 [14] is a protein-guided assembly approach that first clusters the transcripts based on similarity to a common protein and then passes each cluster to CAP3. The recent use of *blast2cap3* on the wheat transcriptome assembly shows that *blast2cap3* generates fewer artificially fused sequences compared to assembling the entire dataset with CAP3. Moreover, it also reduces the total number of transcripts by 8-9% [15].

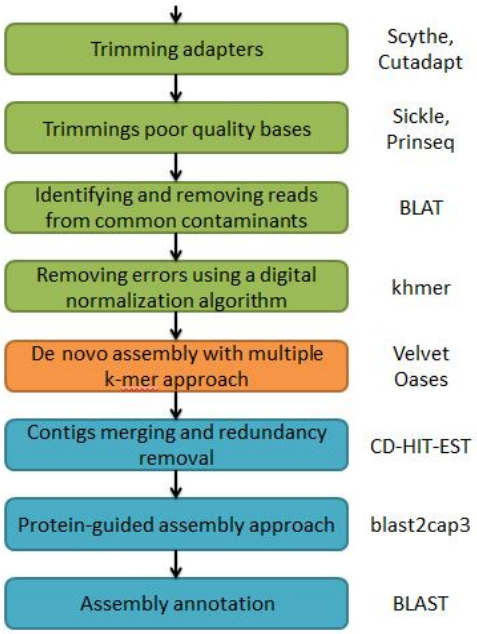


Figure 1. General transcriptome assembly pipeline with some common steps and the tools used for those steps.

Before running *blast2cap3*, the assembled transcripts are aligned with protein datasets closely related to the organism for which the transcripts are generated. BLASTX [16] is used for this alignment. Afterwards, transcripts sharing a common protein hit are merged using CAP3. Therefore, *blast2cap3* uses the assembled transcripts and the BLASTX alignments as an input files.

III. PEGASUS WORKFLOW MANAGEMENT SYSTEM

Pegasus Workflow Management System (Pegasus WMS) stands for Planning for Execution in Grids. Pegasus WMS is a framework that automatically maps high-level scientific workflows organized as directed acyclic graph (DAG) onto wide range of execution platforms, including clusters, grids, and clouds [2]. Pegasus receives an abstract workflow and tries to simplify it before mapping it into a concrete workflow. The abstract workflow of Pegasus contains information and description of all executable files (transformations) and logical names of the input files used by the workflow. On the other hand, the concrete workflow specifies the location of the data and the execution platform [20]. The concrete workflow is then submitted to Condor's DAGMan meta-scheduler [3] for execution [18]. The high-level of abstraction of Pegasus allows scientists to ignore low-level configurations required by the middleware and the underlying execution platform [20].

DAG-based workflows use nodes to define the tasks and use edges to denote the task dependencies. In DAG-based workflows, the structure can be characterized as *sequence* and *parallel* [17]. The sequence structure is defined as an ordered series of tasks, where one task starts after the

previous task is completed. The parallel structure allows concurrently execution of tasks. Pegasus also allows clustering of small tasks into larger clusters that are scheduled and executed to the same remote site. This setting allows improvement of the performance and reducing the remote execution overheads [19].

Pegasus uses DAX (directed acyclic graph in XML) files to specify an abstract workflow. The DAX file contains syntax for defining jobs, arguments, input and output files, and dependencies between the various tasks. This format is shared by many workflow tools. The DAX file can be created manually, or by using the Pegasus API. Pegasus uses Java, Perl, or Python libraries for writing DAX generators [19]. The abstract DAX is then mapped to one or more execution sites. This step is known as the *planning stage*.

Pegasus comes with a set of useful command-line tools that help users to submit and analyze the workflows, and generate useful statistics and plots about the workflow performance, running time, execution results, machines used, as well as for succeeded and failed tasks [19]. *Pegasus-plan* is used to plan the workflow, while *pegasus-run* is used to submit the workflow to DAGMan. After the workflow is submitted, it can be monitored using the *pegasus-status* command that shows information about the running jobs and the percentage of finished jobs. The whole workflow and the failed jobs can be debugged using the *pegasus-analyzer* tool. After the workflow execution ends, the resulting data can be summarized using *pegasus-statistics* and *pegasus-plots*.

Pegasus is used in a number of large scientific applications built for physics, astronomy, biology, earthquake sciences, ocean sciences, limnology and many other domains [20][21][22][23]. Pegasus can use both single systems and heterogeneous set of resources for executing the scientific workflows. The used resources can be distributed across laptops, campus clusters, grids and cloud platforms. Furthermore, Pegasus can support workflows ranging from a few computational tasks to a few millions.

Scalability and handling large sets of data and computations, portability and ease of use are just part of the advantages that Pegasus has. In case of a job or data transfer failure, Pegasus can retry the job or the entire workflow given number of times. If the job fails again, then Pegasus generates a rescue workflow that contains information of the work that remains to be done such that it can be modified and resubmitted later. Therefore, Pegasus has capabilities for provenance tracking, execution monitoring and management, and error recovery.

IV. Execution Platforms

The resources that these scientific workflows require can exceed the capabilities of the local computational resources. Therefore, the scientific workflows are usually executed on distributed platforms, such as campus clusters, grids or clouds. These platforms are usually a set of heterogeneous hosts that are connected via a network. The host that is able to schedule remote jobs and has the appropriate software for execution of these jobs is known as a *submit host*. The *submit host* also maintains an information about the remote hosts and the software installed there, and serves for debugging purposes.

A. University of Nebraska Campus Cluster

A campus cluster is a campus wide resource in a university that allows faculty and students to use the resources of the cluster for their computational needs.

Campus clusters may not be highly I/O friendly. Moreover, campus clusters are not instantly available, and thus there is a long waiting time to access nodes with required memory and time resources.

Sandhills is one of the High Performance Computing (HPC) Clusters at the University of Nebraska-Lincoln Holland Computing Center (HCC) [24]. Sandhills was acquired by combining grants from various research groups at University of Nebraska. It is used by faculty and students in disciplines like bioinformatics, nanoscale chemistry, subatomic physics, meteorology, genomics, crashworthiness and artificial intelligence. Sandhills was constructed in 2011 and it has 1,440 AMD cores housed in a total of 44 nodes. Each node has storage of approximately 1.5 TB. Sandhills is a heterogeneous cluster in terms of individual node resources.

Every new user account of HCC is required to be associated with a faculty or research group. The allocation of computing resources at HCC is done on group basis where the group owner has ownership of all files in the group account.

B. Open Science Grid (OSG)

The Open Science Grid (OSG) is a national consortium of geographically distributed academic institutions and laboratories that provide hundreds computing and storage resources to the OSG users.

The OSG is organized into Virtual Organizations (VO's) which include not only the people from an academic community, but also their services, software and policies [25]. OSG does not own any computing or storage resources, but allows users to use the resources contributed by the other members of the OSG and VO's.

Every new user of OSG first needs to apply for an OSG certificate. This step helps sites identify users and their VO's. Once the certificate is verified and approved, the user can import it in the Web Browser. Furthermore, the user requests membership in the community group in which the user belongs by using VOMS (Virtual Organization Membership Service). After this request for registration is approved by one of the community VOMS admins, the user can use the OSG resources.

V. Experiments

In this paper, our objective is to evaluate the performance of a built scientific workflow for protein-guided assembly on a campus cluster and OSG.

The experiment for this paper includes creating and running a scientific workflow for *blast2cap3*, the protein-guided assembly. The workflow is run on two different execution platforms: Sandhills, the campus cluster, and the

OSG. Furthermore, the influence of the number of clusters of transcripts in *blast2cap3* over the execution time is also investigated and compared.

A. Experimental Data

For this experiment, we created an assembly pipeline with the steps shown on Fig. 1 using diploid wheat *Triticum urartu* dataset. The NCBI BioProject PRJNA191053 [26] contains all the sequence libraries submitted by UCD group.

The description given on NCBI for this library construction and sequencing is as follows: "*The sequencing libraries were prepared from shoot and root tissues harvested from 2-3 week old seedlings. All sequencing was carried out on the Illumina HiSeq platform. All libraries were sequenced using the 100 bp paired-end protocol on four lanes of Illumina HiSeq2000 machines at the University of California Davis (UCD) Genome Center. Base quality calls and demultiplexing was done with the CASAVA 1.8.0 pipeline (Illumina).*" [26].

The generated assembly after transcripts merging and redundancy removal, "*transcripts.fasta*", is 404 MB big and contains 236,529 transcripts. In order to use *blast2cap3*, the protein-guided assembly, the next step is to align the transcripts with protein datasets closely related to the wheat [15] using BLASTX. The BLASTX tabular output, "*alignments.out*", is 155 MB big and contains 1,717,454 protein hits.

B. Current Implementation of blast2cap3

Blast2cap3, the protein-guided assembly, is a Python script written by Vince Buffalo [14]. Beside Python's modules [27], *blast2cap3* also uses Biopython, a set of available tools for biological computation written in Python [28], and CAP3 [13].

The current implementation of *blast2cap3* supports only serial execution. This means that first one cluster of similar transcripts is created and then is sent to CAP3. After the CAP3 program terminates, this process is repeated consecutively for all possible clusters of transcripts.

When the existing implementation of *blast2cap3* was run on Sandhills for the given input files "*transcripts.fasta*" and "*alignments.out*" with size of 404 MB and 155 MB respectively, the running time was 100 hours. Considering larger input files and datasets, the time requirements and complexity of running the protein-guided assembly grow.

Each cluster of transcripts that is generated from *blast2cap3* and uses CAP3 is an individual process. This means that as long as the final results from CAP3 for each cluster are concatenated at the end, the transcripts within the cluster can be generated and merged independently.

Therefore, an additional approach to *blast2cap3* execution should be considered that requires not just a single computer, but multiple computational nodes that will use the modularity of *blast2cap3* execution.

C. Pegasus Workflow Management System Implementation of blast2cap3 for Sandhills

The modularity of *blast2cap3* allows us to decompose the existing approach on multiple tasks, some of which can be run in parallel. Therefore, this protein-guided assembly can be structured into a scientific workflow using the Pegasus Workflow Management System. The main reduction in the running time of the current implementation of *blast2cap3* is expected to be reached when the merging of transcripts belonging in a cluster is done in parallel for all clusters.

The Pegasus WMS implementation of *blast2cap3* for Sandhills is shown on Fig. 2.

For this workflow, we first create lists of both input files, "*transcripts.fasta*" and "*alignments.out*", respectively. These two tasks are independent of each other, and can be run at the same time. Furthermore, in order to create multiple clusters of transcripts, the *split()* task is used to divide the big "*alignments.out*" file on "*n*" smaller files. For the purpose of this paper, we use different values of "*n*", such as 10, 100, 300, and 500.

The number of tasks that merge the transcripts within a cluster depends on "*n*", the number of clusters. From the workflow shown on Fig. 2, we can notice that this task, *run_cap3()*, uses two input files, "*transcripts_dict.txt*" and "*protein_n.txt*".

After "*n*" output files are generated from *run_cap3()*, the next step is to merge all these joined transcripts into one file. Knowing the transcripts that are joined helps us to combine all transcripts that are not joined into a new file.

The DAG structure of the workflow is helpful to define dependencies, and execute a task if and only if its predecessor tasks have finished.

D. Pegasus Workflow Management System Implementation of blast2cap3 for OSG

The Pegasus WMS implementation of *blast2cap3* for OSG is shown on Fig. 3. The workflow and the logic behind both execution platforms differ only in the way how certain tasks are defined. The resources provided by Sandhills, the campus cluster, contain the most frequently used libraries, modules and software tools. This means that the Python and Biopython libraries and the CAP3 executable required by *blast2cap3* are already set and maintained on the campus cluster. On the other hand, the resources provided by OSG are more heterogeneous and most of the time belong to other academic institutions and laboratories that may provide different software and system configurations.

When the required libraries and executables like Python, Biopython and CAP3 are not installed on the remote node, the workflow execution fails. In order to avoid workflow failures, additional tasks that download and install the necessary software are executed before the main tasks in the workflow. These modified tasks are represented with red rectangles on Fig. 3.

Therefore, we can say that the Pegasus WMS implementation of *blast2cap3* for OSG is a slightly modified version of the implementation of *blast2cap3* for Sandhills.

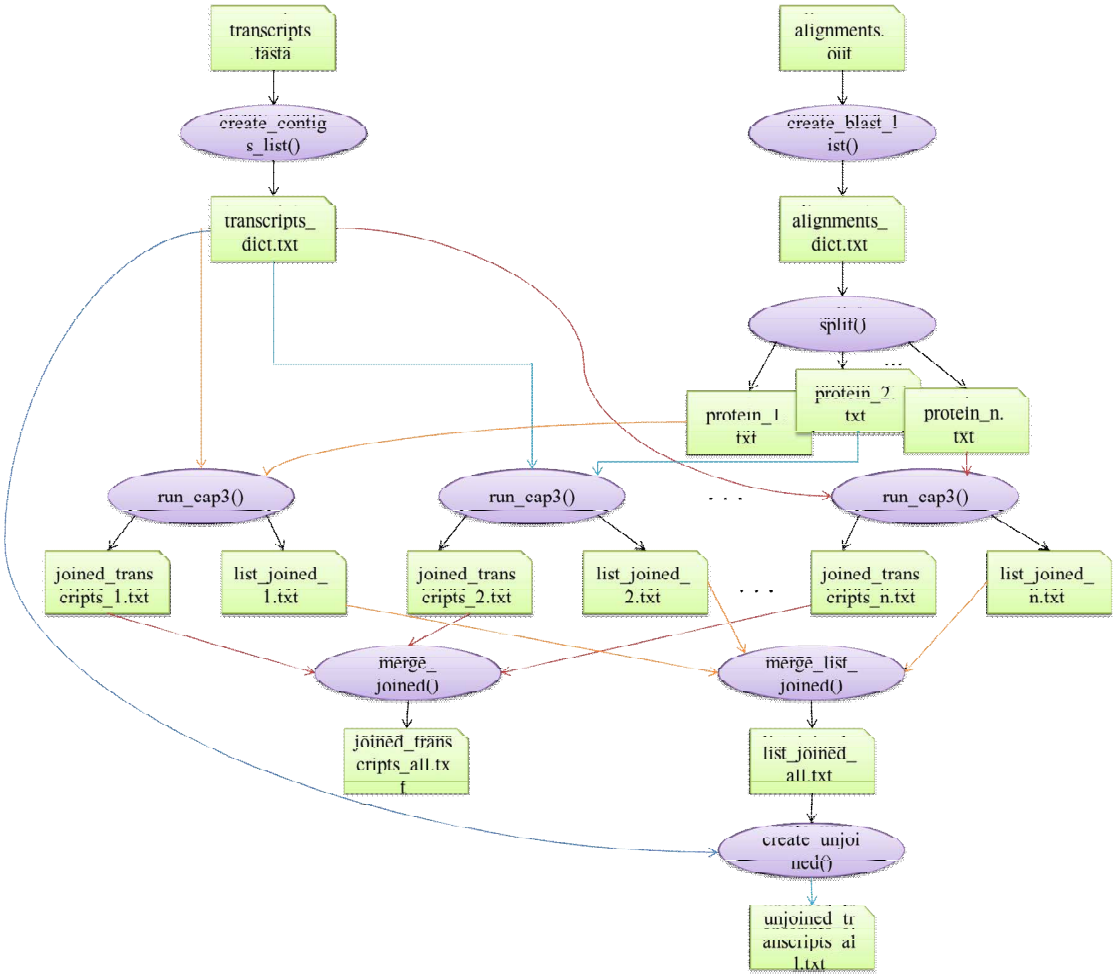


Figure 2. Pegasus WMS implementation of *blast2cap3* for Sandhills, where the squares represent the input and output files, the ovals represent the tasks, and the arrows represent the dependencies between the tasks.

VI. Performance Evaluation

After the scientific workflow was created using Pegasus WMS, it was run on each platform multiple times with different values for "*n*". As mentioned previously, "*n*" determines the number of clusters of transcripts on which the input data, "*alignments.out*", is divided. For the purpose of this paper, we used "*n*" with values of 10, 100, 300, and 500.

A. Comparing Running Time on Sandhills and OSG for Different Values of "n"

In order to compare the running time of the Pegasus WMS implementation of *blast2cap3*, we run the workflows when "*n*" is 10, 100, 300, and 500 respectively. After the workflow terminates, *pegasus-statistics* is used to generate general statistics for the workflow execution. We use these statistics to compare the running time when *blast2cap3* is run serially and when is run as a scientific workflow with different values of "*n*".

The "*Workflow Wall Time*" statistic defines the total running time of the workflow from the start to its end. The comparison of this variable's value for the different workflows executed on the different platforms is shown on Fig. 4.

On Fig. 4 we can notice that the Pegasus WMS implementation of *blast2cap3* significantly reduces the time execution for approximately more than 95%. If the current sequential implementation of *blast2cap3* for the given input files runs for 100 hours, the Pegasus WMS implementation runs for 3 hours in average.

Beside the difference between the serial and inherently parallel execution of *blast2cap3*, on Fig. 4 we can also observe the difference in the running time on Sandhills and OSG platforms.

Although OSG provides bigger variety of computational resources than Sandhills, for the experimental runs of our workflows, Sandhills resulted in better running time. This difference is especially noticeable when "*n*", the number of clusters used, is 10, 100, and 300.

Some possible reasons for this occurrence are the additional tasks required for setting the proper software configuration on the OSG resources, as well as the common

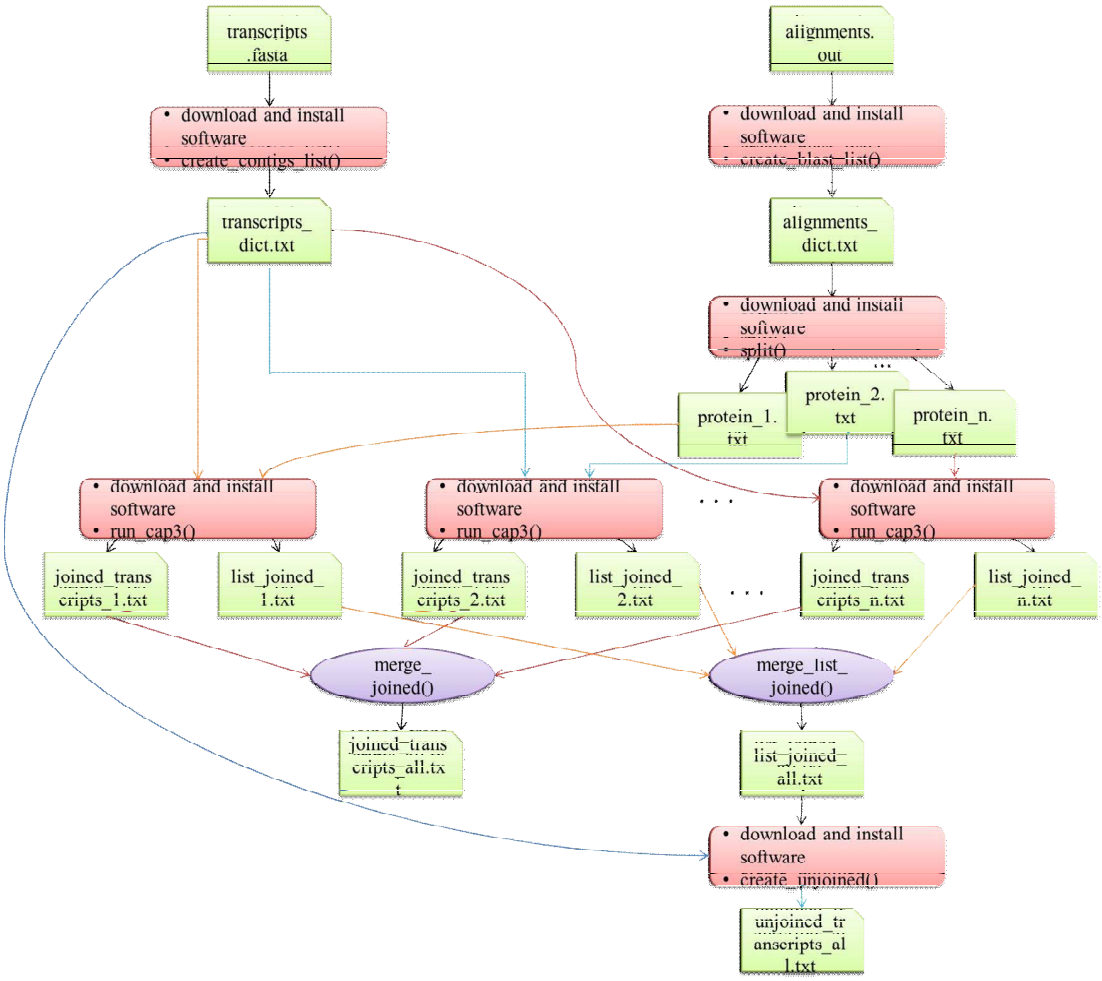


Figure 3. Pegasus WMS implementation of *blast2cap3* for OSG, where the squares represent the input and output files, the ovals represent the tasks, the rectangles represent the tasks that has an additional step of downloading and installing the required libraries, and the arrows represent the dependencies between the tasks.

failures and workflow retries that happen when OSG is used as a platform. The OSG users use the resources that belong to other VO groups, and if the members of that group submit jobs in meanwhile, the OSG user job may be cancelled or held. On the other hand, we encountered no failures when the workflow was executed on Sandhills. The campus cluster may need a long waiting time to access nodes with required memory and time resources, but after these resources are allocated, they are utilized until the tasks terminate.

The running time on Sandhills when "*n*" is 10 is 41,593 seconds. On the other hand, when "*n*" has value of 100, 300, and 500, the running time on Sandhills is around 10,000 seconds. The usage of 100 or more clusters of transcripts improves the running time on Sandhills for approximately 80% compared to the running time of 10 clusters. Although the usage of more than 100 clusters doesn't decrease this running time significantly, the selection of 300 clusters gives the optimum performance with the resources allocated from Sandhills for this experiment. We must emphasize that the running time for the both platforms and the optimal number of used clusters of transcripts may vary for every new run due to the availability of the current resources.

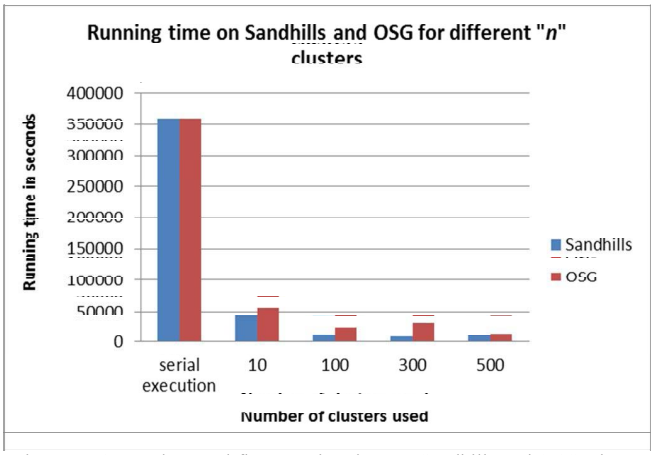


Figure 4. Comparing workflow running time on Sandhills and OSG when *blast2cap3* is executed serially and as a scientific workflow with "*n*" is 10, 100, 300, and 500 respectively.

B. *Comparing Running Time Per Task on Sandhills and OSG for Different Values of "n"*

The running time of the submitted tasks and jobs varies among the two execution platforms and "*n*", the number of clusters of transcripts. In addition of this Section, we will analyze the running time of the individual tasks from the workflow, both for Sandhills and OSG when "*n*" is 10, 100, 300, and 500.

In order to achieve this, we use "*Kickstart Time*", "*Waiting Time*" and "*Download/Install Time*" statistics.

The "*Kickstart Time*" statistic defines the actual duration and running time of a job on the remote node.

The "*Waiting Time*" statistic is a sum of the time spent waiting on the *submit host* and the time spent waiting on the remote host before the actual execution starts.

The "*Download/Install Time*" statistic refers to the Pegasus WMS implementation of *blast2cap3* for OSG and indicates the time spent for downloading and installing the Python and Biopython libraries and CAP3 executable required for this experiment.

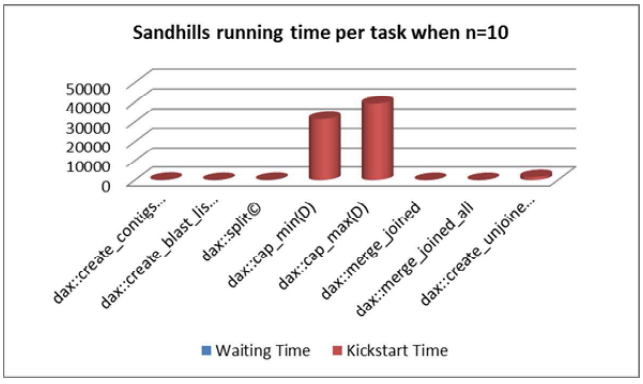
On Fig. 5 the running times per tasks are shown for both Sandhills and OSG execution platforms when "*n*" is 10, 100, 300, and 500 respectively.

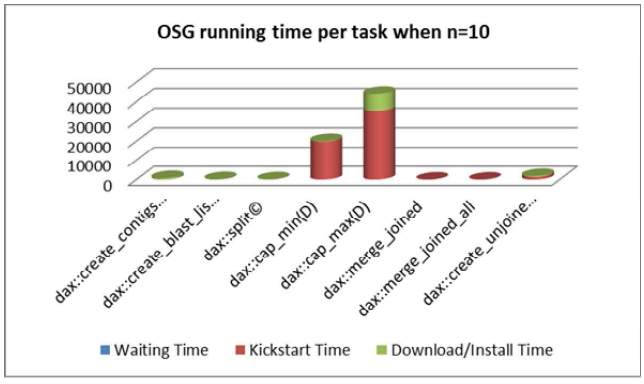
While the tasks for creating lists of the input files and for merging the final results have running time of few minutes, the higher consumption of time occurs when CAP3 is used for merging the transcripts within the clusters.

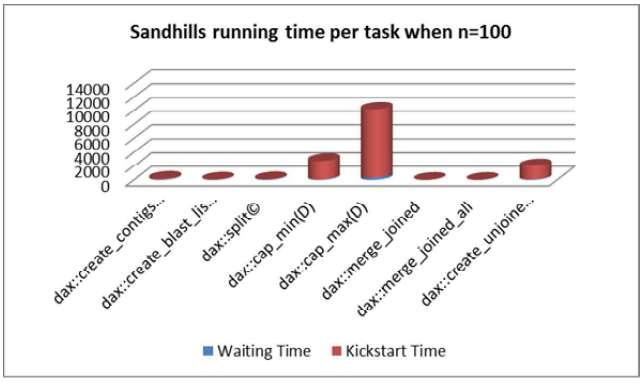
The "*Waiting Time*" value for the tasks ran on Sandhills is small and negligible. On the other hand, this value unevenly changes, increases and decreases, for the tasks ran on OSG. This observation once again shows that the resources available on OSG are opportunistic, and the OSG user can not control the availability or the lack of resources over time. Unlike Sandhills, failures and retries of the workflow were observed on OSG. This occurrence that is generally common and frequent on grids also increases the value of the "*Waiting Time*" statistic.

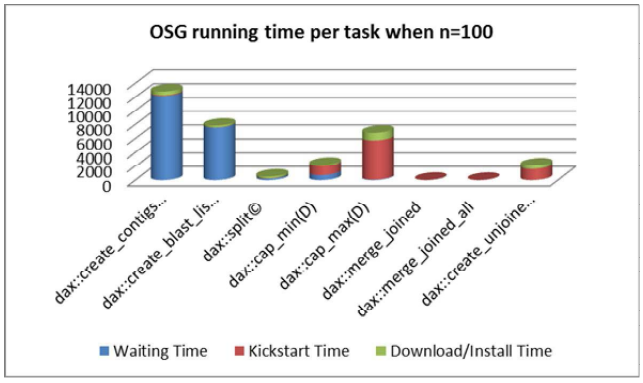
The "*Kickstart Time*" value per task on Sandhills slowly decreases when "*n*" increases. Higher values of "*n*" induce even more significantly greater reduction of the running time of the tasks ran on OSG.

However, the "*Download/Install Time*" value influences over the total running time of the tasks ran on OSG. Although some tasks on OSG have smaller running time than the tasks ran on Sandhills for the same value of "*n*", they still exceed the running time of the tasks on Sandhills.









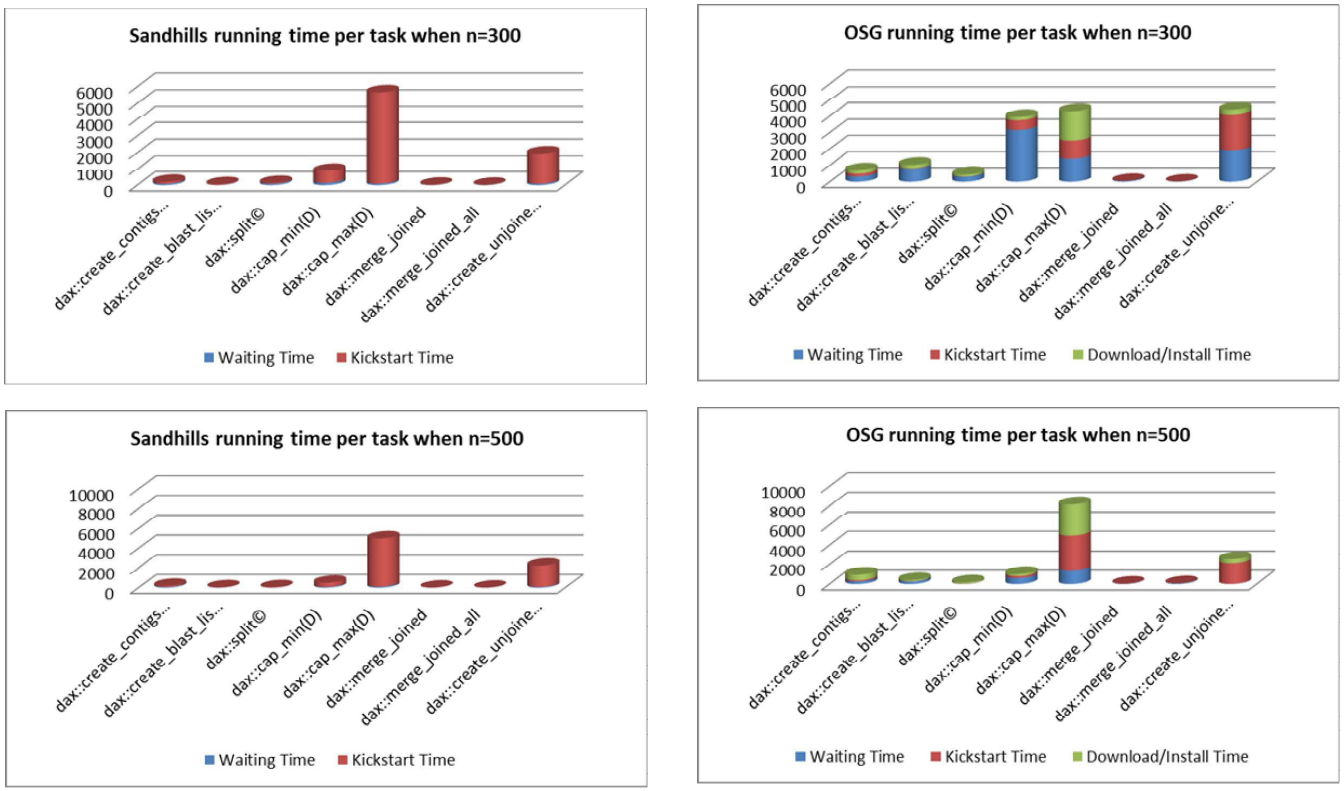


Figure 5. Comparing *blast2cap3* workflow running time per task on Sandhills and OSG when "*n*" is 10, 100, 300, and 500 respectively.

This happens because an additional time is required for the tasks on OSG to download and install the necessary libraries and executables on the OSG resources.

VII. CONCLUSION

The expansion of scientific data leads to research that requires complex and data-intensive analyses and simulations. Therefore, many scientists use workflows over distributed resources to manage these large and complex computational tasks. Workflow applications can be used in different scientific fields, such as biology, physics, astronomy, and many others.

In this paper we build a scientific workflow for *blast2cap3*, the protein-guided assembly, using the Pegasus Workflow Management System (Pegasus WMS). Furthermore, we describe our experience deploying this workflow on two different distributed execution platforms: Sandhills, the University of Nebraska Campus Cluster, and the Open Science Grid (OSG). Our objective was to compare and evaluate the performance of the built scientific workflow for both platforms used. Furthermore, we wanted to show the importance of using scientific workflows for executing computationally demanding granular tasks and pipelines.

The performed experiments for this paper show that the Pegasus WMS implementation of *blast2cap3* ran on both platforms significantly reduces the running time of the current serial implementation of *blast2cap3* for more than 95 %. This high percentage shows the importance and the efficiency of using scientific workflows.

Beside the difference between the serial and parallel execution of *blast2cap3*, we also observed the difference in the running times on both Sandhills and OSG execution platforms. Although OSG provides bigger variety of computational resources than Sandhills, for our experiments, the workflows ran on the campus cluster resulted in better running time. Moreover, the selection of 300 clusters of transcripts gives the optimum performance with the resources allocated from Sandhills for the completed experiment.

While the Sandhills resources contain the most frequently used software tools, the OSG resources may have different software configuration. Therefore, the tasks on OSG used more running time than the tasks running on Sandhills because of downloading and installing the required libraries and tools for *blast2cap3*. In addition, the availability of resources on OSG is highly variable and opportunistic, and therefore the performance and the running time of the tasks

vary significantly. Workflows running on OSG may result with excellent or very poor results depending whether there are plenty or few available resources. In addition, workflow failures and retries were observed on OSG that also increase the running time.

However, if comparing only the actual duration and running time of tasks on both platforms, ignoring the "*Waiting Time*" and the "*Download/Install Time*", OSG gives significantly better results. Hence, setting the proper software configuration on the OSG resources for less time will be considered as part of the future work.

Despite campus clusters and grids, recently the scientists are investigating the use of clouds for deploying scientific workflows. Using academic and commercial clouds as an execution platform for the *blast2cap3* workflow built in this paper will be challenging, but important and useful further step of this research.

Developing scientific workflows for applications from different scientific fields is a valuable and crucial step that connects complex and large granular tasks with thousands available powerful computational and distributed resources. The outcome of this process are automated complex analysis, real-time results and improved time performance that allow scientists to easily design, execute, modify and re-run their experiments.

ACKNOWLEDGMENT

We would like to thank Dr. Adam Caprez, HPC Applications Specialist at the University of Nebraska Holland Computing Center, for his extensive help and useful suggestions during the preparation of the experiments for this paper.

This work was completed utilizing the Holland Computing Center of the University of Nebraska, as well as using resources provided by the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science.

REFERENCES

[1] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers, "Examining the Challenges of Scientific Workflows," Computer 40.12 (2007): 24-32.

[2] E. Deelman, G. Singha, M. Sua, J. Blythea, Y. Gila, C. Kesselmana, G. Mehtaa, K. Vahia, G. Berrimanb, J. Goodb, A. Laityb, J. Jacobc, D. Katzc, "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems," Scientific Programming Journal, Vol 13(3), pages 219-237, 2005.

[3] P. Couvares, T. Kosar, A. Roy, Jeff Weber and Kent Wenger, "Workflow in Condor", in In Workflows for e-Science, Editors: I.Taylor, E.Deelman, D.Gannon, M.Shields, Springer Press, January 2007 (ISBN: 1-84628-519-4).

[4] D. Thain, T. Tannenbaum, M. Livny, "Distributed Computing in Practice: The Condor Experience," Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[5] T. Oinn, M. Greenwood, M. Addis, M Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences," Concurrency Computat: Pract. Exper., 18: 1067–1100 (2006). doi: 10.1002/cpe.993.

[6] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, Y. Zhao, "Scientific workflow management and the Kepler system," Concurrency Computat.: Pract. Exper., 18: 1039–1065, (2006). doi: 10.1002/cpe.994.

[7] S. Cohen-Boulakia, U. Leser, "Search, Adapt, and Reuse: The Future of Scientific Workflow," SIGMOD Record 40(2):6-16, 2011.

[8] Open Science Grid. [http://www.opensciencegrid.org/].

[9] Extreme Science and Engineering Discovery Environment (XSEDE). [http://www.xsede.org/].

[10] Amazon Elastic Compute Cloud. [http://aws.amazon.com/ec2/].

[11] FutureGrid. [http://futuregrid.org/].

[12] Y. Surget-Groba, J. Montoya-Burgos, "Optimization of de novo transcriptome assembly from next-generation sequencing data," Genome Res. 2010 Oct;20(10):1432-40. doi: 10.1101/gr.103846.109.

[13] H. Xiaoqiu, M. Anup, "CAP3: A DNA Sequence Assembly Program," Genome Res. 1999 September; 9(9): 868–877.

[14] Buffalo V: *Blast2cap3* software . [https://github.com/vsbuffalo/blast2cap3/].

[15] K. Krasileva, V. Buffalo, P. Bailey, S. Pearce, S. Ayling, F. Tabbita, M. Soria, S. Wang, IWGS Consortium, E. Akhunov, C. Uauy, J. Dubcovsky, "Separating homeologs by phasing in the tetraploid wheat transcriptome," Genome Biology 2013, 14:R66 doi:10.1186/gb-2013-14-6-r66.

[16] S. Altschul, W. Gish, W. Miller, E. Myers, D. Lipman, "Basic local alignment search tool, J Mol Biol 1990, 215:403-410.

[17] J. Yu, R. Buyya, "A taxonomy of scientific workflow systems for grid computing," SIGMOD Rec., vol. 34, pages 44–49, September 2005, 2, 13, 56.

[18] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," Journal of Grid Computing 2003, Volume 1, Issue 1, pp 25-39.

[19] Pegasus 4.3 User Guide. [https://pegasus.isi.edu/wms/docs/latest/pegasus-user-guide.pdf/].

[20] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, "Pegasus: Planning for Execution in Grids," GriPhyN technical report 20(17):12-22

[21] J. C. Good, J. C. Jacob, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, M. Su, and R. Williams, "Astronomical Image Mosaicking on a Grid: Initial Experiences," in Engineering the Grid: Status and Perspective, B. D.Martino, J. Dongarra, A. Hoisie, L. T. Yang, and H. Zima, Eds.: American Scientific Publishers, 2006.

[22] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman, "Workflow Management in GriPhyN," in Grid Resource Management: State of the Art and Future Trends, J. Nabrzyski, J. M. Schopf, and J. Weglarz, Eds.: Springer, 2003.

[23] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R.Graves, N. Gupta, V. Gupta, T. H. Jordan, C. Kesselman,P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi,and L. Zhao, "Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example," presented at Second IEEE International Conference on e-Science and Grid Computing, 2006.

[24] Sandhills UNL HPC Cluster. [http://hcc.unl.edu/sandhills/].

[25] M. Altunay, P. Avery, K. Blackburn, B. Bockelman, M. Ernst, D. Draser, R. Quick, R. Gardner, S. Goasguen, T. Levshina, M. Livny, J. McGee, D. Olson, R. Pordes, M. Potekhin, A. Rana, A. Roy, C. Sehgal, I. Sfiligoi, F. Wuerthwein, The Open Science Grid Executive Board, "A Science Driven Production Cyberinfrastructure - the Open Science Grid," Journal of Grid Computing (Impact Factor: 1.6). 9(2):201-218. DOI:10.1007/s10723-010-9176-6 Source: dx.doi.org.

[26] NCBI BioProjects [http://www.ncbi.nlm.nih.gov/bioproject/?term=PRJNA191053/]/

[27] Python Programming Language. [http://www.python.org/].

[28] Biopython. [http://biopython.org/].

[29] R. Littauer, K. Ram, B. Ludascher, W. Michener, R. Koskela, "Trends in Use of Scientific Workflows: Insights from a Public Repository and Recommendations for Best Practice," The International Journal of Digital Curation, Volume 7, Issue 2 | 2012.

[30] H. AlHakami, H. Aldabbas, T. Alwada'n, "Comparison Between Cloud and Grid Computing: Review Paper," International Journal on Cloud Computing: Services and Architecture (IJCCSA),Vol.2, No.4, August 2012.