World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:7, No:2, 2013

Modular Workflow System for HPC Applications

Y. Yudin, T. Krasikova, Y. Dorozhko and N. Currle-Linde

***Abstract*—**Nowadays, HPC, Grid and Cloud systems are evolving very rapidly. However, the development of infrastructure solutions related to HPC is lagging behind. While the existing infrastructure is sufficient for simple cases, many computational problems have more complex requirements.Such computational experiments use different resources simultaneously to start a large number of computational jobs.These resources are heterogeneous. They have different purposes, architectures, performance and used software.Users need a convenient tool that allows to describe and to run complex computational experiments under conditions of HPC environment.

This paper introduces a modularworkflow system called SEGL which makes it possible to run complex computational experiments under conditions of a real HPC organization. The system can be used in a great number of organizations, which provide HPC power. Significant requirements to this system are high efficiency and interoperability with the existing HPC infrastructure of the organization without any changes.

***Keywords*—**HPC, Molecular Dynamics, Workflow Languages, Workflow Management.

I. Introduction

EACHHPC organization that provides computational power has its own scheme of resource management, security policies, concepts of access rights, limitations for the use of computational resources, disk space, memory and network. Organization may offer different cost models for different purposes of resource usage, such as commercial usage or scientific research usage. Resources can be provided free of charge or as a paid service. Accordingly, user jobs will have different priorities and limitations.

However, there are a lot of common or similar principles in the existing frameworks to organize work with a HPC resource in a real organization. Commonly, these principles are based on widespread and well-known open-source software. Usually resources run Linux/Unix-like operating systems (Linux/Unix OS are installed on 95% of the HPC computers in the TOP500 [1]). Most computational resources, clusters or supercomputers, have one or more access-point machines (so called front-ends). A job submission is organized with the help of batch systems [2] in accordance with queue policies, priorities and limitations existing in the organization.

Users can access front-end machines within the local network of the organization via SSH [3]. Access from an external network can be denied or can be restricted and provided only for fixed IP addresses. Quite often resources are accessed from an external network through VPN (Virtual Private Network). Users have their own accounts on all accessible resources of the organization. Data exchange between users and resources is organized using SSH [3] and such well-known programs as SCP, SFTP, RSYNC.

The approach described above is widespread in organizations which offer HPC resources, because of simple installation and support, high reliability and security.

The standard scenario for working with HPC resources is the following: the user transfers input data to the selected resource. Then the user submits a job to the queue. After that the user has to check state of the job periodically. Often such a job is part of a complex computational task. A large number of computational jobs can be started within the scope of the task in parallel or sequentially. Jobs can be dependent on other jobs, if some resulting data of one job is used as an input data for another job. A user, who starts such tasks (or experiments) requires a suitable tool to create the computational experiment, to run it and to monitor it. Workflow systems [4] for HPC environments fit these requirements.

In this paper we describe our experience in development and usage of the high performance workflow system SEGL (Science Experimental Grid Laboratory). This system is designed to work in a real HPC organization.The key goal of our work was to develop a comprehensive platform to describe, run and analyze complex large-scale data-intensive simulation experiments. For this new methods for describing such complex scalable experiments – based on GriCoL (Grid Concurrent Language) [5], [6]– were developed. Furthermore, a specific middleware solution supporting data-intensive tasks was developed and implemented. Finally, a graphical user interface was implemented which provides the necessary tools for the whole simulation experiment i.e. model description, simulation running, simulation monitoring and data processing. One of the key requirements was to get a fully functional, efficient, and user-friendly system which provides not only some testing mode for scientific experiments, but enables a full-blown efficient cooperation between end users and the system in production mode.

The main requirements to the SEGL system were the following:

Y. Yudin is with High Performance Computing Center Stuttgart (HLRS), University of Stuttgart,Nobelstr 19,70597 Stuttgart, Germany (phone: +49-711-685-87263; fax: +49-711-685-87209; e-mail: yudin@hlrs.de).

T. Krasikova is with High Performance Computing Center Stuttgart (HLRS), University of Stuttgart, Nobelstr 19, 70597 Stuttgart, Germany (phone: +49-711-685-87263; fax: +49-711-685-87209; e-mail: krasikova@hlrs.de).

Y. Dorozhko is with High Performance Computing Center Stuttgart (HLRS), University of Stuttgart, Nobelstr 19, 70597 Stuttgart, Germany (phone: +49-711-685-87216; fax: +49-711-685-87209; e-mail: dorozhko@hlrs.de).

N. Currle-Linde is with High Performance Computing Center Stuttgart (HLRS), University of Stuttgart, Nobelstr 19, 70597 Stuttgart, Germany (phone: +49-711-685-65801; fax: +49-711-685-87209; e-mail: linde@hlrs.de).

- The system must work under conditions of the real HPC organization without any changes of the infrastructure of the organization.
- Reliability of the system.
- Simplicity of using HPC resources by applied specialists (scientists, engineers) with the help of this system.
- Ability to reuse existing code. This means reusing of the predefined computational jobs as well as the complex computational experiments.
- Scalability of the system to complex simulation experiments that consist of large number of computational tasks (up to tens of thousands and hundreds of thousands).

All these requirements have the same priority.

It should be noted that we are talking about real HPC organization, but not about a virtual organization (VO)[7]. There are a number of differences between a VO and a real HPC organization [8] concerning HPC resources from the viewpoint of the workflow developer. There are such questions as administration of the resources, network communications and security. Often computational resources of the real organization are used as resources of the VO, but resources of the VO can have different settings and limitations. Also user support in a VO is a separate and complicated problem[9].

However in the case of using the HPC resources for business and industrial tasks foreground questions are security, commercial advantage, quality of user support and responsibility of the organization that provides resources. Using resources as a part of the VO supposes some changes of the infrastructure of the real organization. We solve the problem by developing a workflow system within the scope of the real HPC organization, without any infrastructure changes.

In this article we start with a brief overview of the related work in this field. Then we introduce the workflow description languageGriCoL. In the fourth, fifth and sixth sections we present the executive environment and the middleware solutions of the SEGL system. Molecular Dynamics simulation is described in the seventh section. This is a good example for a complex simulation workflow. Furthermore it shows that our concept can be successfully used in a true production environment.

II. Related Work

The development of simulation technologies includes various scientific branches. One of the key strategies is creating integrated simulation environments, which would support all aspects of creating and executing simulation applications. These requirements are partially met by “Scientific Workflows” and “Science Gateways”.

Currently Science Gateways are based on portal technologies and often include different possibilities for creating and running complex applications. One such system is the P-Grade Portal [4]. This is a web based, service rich environment for the development, execution and monitoring of workflows and workflow based parameter studies on various grid platforms. P-Grade can support both task- and service-based workflows and provides interoperability with Globus Toolkit, LCG and gLiteGrid middleware.

General purpose open source workflow systems include Taverna, Kepler, and Triana[4], [10]. A service-based approach has been adoptedin all these well-known workflow managers. Developed for supporting scientific workflows these systems have a very similar approach: the workflow is visually constructed from components, which can be local processes, tasks for grid resources or can invoke remote services such as Web services. These workflow systems can interact with Grid resources via middleware systems, such as Globus Toolkit, gLite, Unicore. Initially these systems were not intended to execute data-intensive workflows.

Other systems, such as the LONI Pipeline for neuro-imaging and the commercial Pipeline Pilot for drug discovery, are more geared towards specific applications and are optimized to support specific component libraries [11].

Obviously, one of the major tendencies in the field of simulation is the rapid growth of data volumes which happens simultaneously with the development of HPC technologies. This leads to new applications which have to deal with thousands of data sets. Speaking about examples of large scale data-intensive workflow systems [12]Pegasus[4], [10] is a good example.Pegasus is a framework that allows mapping of workflow instances onto a set of distributed resources such as the Grid or a Condor pool. The mapping process not only involves finding the appropriate resources for the tasks but also may include some workflow restructuring geared toward improving the performance of the overall workflow. Scalability challenges in the workflow description, management, and performance analysis and the solutions can be illustrated by the usage of Pegasus in earthquake science application [12]. The complicacy of this experiment consisted in the big number of data sets, as Pegasus was able to cope with it, it satisfied the requirements ideally. It supports runtime vertical and horizontal job clustering to reduce scheduling overheads [13].

One more data-intensive workflow management system is ASKALON [14]. In ASKALON, the user composes a Grid workflow application graphically using a UML-based workflow composition and modeling service. To optimize execution of data-intensive applications in ASKALON algorithms based on performance predictions are used. A performance prediction service estimates execution times of workflow activities through a training phase and statistical [15].

External representations for workflow instances, whether they are based on control flow or data flow, are often very similar. One of the most common forms is that of a directed acyclic or cyclic graph (DAG or DCG) with nodes and vertices. Petri Nets are another popular representation method for workflow. Other representations include scripting languages that model the relationships between tasks as a series of ordered function call, and Unified Modeling Language (UML) diagrams that use the standard diagrams and representations to model the relationships [4]. The general-purpose workflow language BPEL is also used to create

executable simulation workflows[6].Although there are many languages to describe workflows, none of them is particularly designed for data-intensive workflows with dynamic nature of data sets behavior.

Conclusion: there is a wide variety of systems which can support complex applications and represent an integrating environment for experiments from different spheres of scientific fields. However, it is difficult to figure out one solution which would be a convenient platform for supporting the whole cycle of the users' activities in different science areas on the one hand, and a highly-effective adaptive solution in the field of execution and post -processing of complex data-intensive applications and simulations, on the other hand. Thus, the motivation of our work is to implement all parts of an efficient platform for complex computational tasks, which operate at the "triple point" of dynamic, distributed and data-intensive (3D) attributes.

III. Workflow Description Language

The Grid Concurrent Language (GriCoL) [16]is used in the system to describe a computational experiment as a workflow.

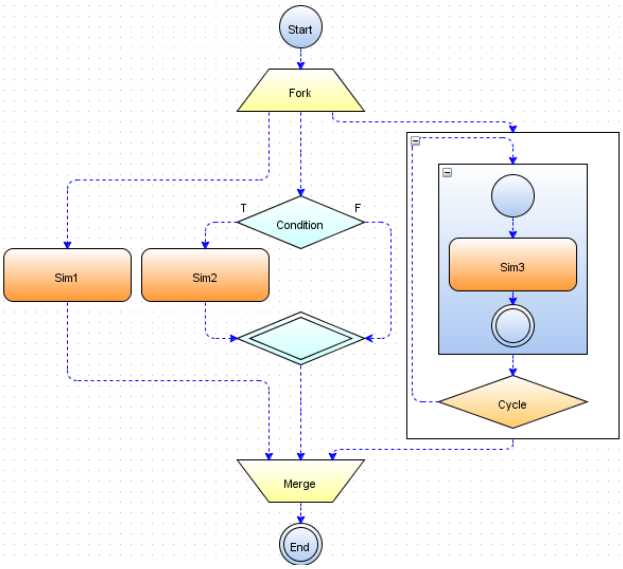

Fig. 1 Control flow level

GriCoL has a two-layer model for the description of the experiment. There are the control flow level and the data flow level.

The control flow level (Fig. 1)offers possibilities to describe set of parallel and sequential tasks within the scope of the experiment. This description is the workflow of the experiment. To describe the logic of the experiment, the control flow level offers different types of blocks: solver, transition, fork/merge (parallel branching), cycle, conditional transition, nested experiment.

For each computational block in the control flow level the user has to describe data flow model.

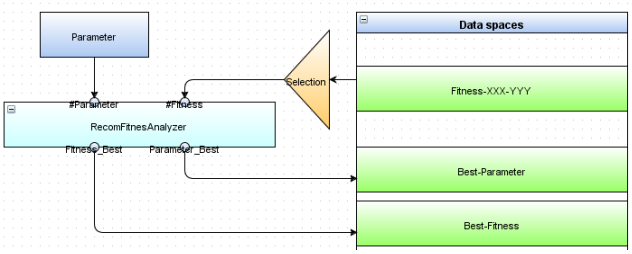

Fig. 2 Data flow level

The data flow level (Fig. 2) description contains detail information about the computational job that will be started on the HPC resource. Also this information includes descriptions of input and output data sets, variables that describe different parallel data sets, parameter values for the different data sets and expressions to select and group the data sets.

GriCoL provides a modular approach[17]. Jobs that will be started on the HPC resource are described in the system as executable modules. A module is an abstract description of the computational job. It includes information about input and output data sets. The module description is used to create a computational experiment. Each executable module has one or more implementations. Different implementations for different resources can be used. In runtime the system selects the most suitable resource for each job. Then system runs appropriate implementation of the executable module on the selected resource.

The system keeps a library of described executable modules and a number of implementations for each module. This allows to reuse an executable module in different experiments. The set of the module implementations can be changed in the case of changes in the set of computational resources.

Once described a computational experiment can be run many times. Description of the input and output data sets does not refer to file and directory names. This allows to run experiments with different input data sets, which have different file names and different numbers of data sets. GriCoL offers its own language elements to describe data communication in the experiment. Such description weakly depends on real runtime data and number of data sets. This approach allows to run a once described experiment with an undefined number of data sets. Also data sets can be parameterized with a set of defined variables.

GriCoL supports the description of nested experiments. The number of nesting levels is unlimited. This allows to use simple experiments as a part of a more complicated experiment.

IV. Workflow Engine Implementation

The workflow engine is a part of a SEGL system server. The system server is installed in the local network of the organization. It has the following tasks: running of the workflow engine, communication with system agents to run computational jobs and handling of client requests. The server keeps all required information in a relational database. This is information about such entities as computational resources of

the organization, executable modules, implementations of the executable modules, experiment models and system users (see Fig. 3).

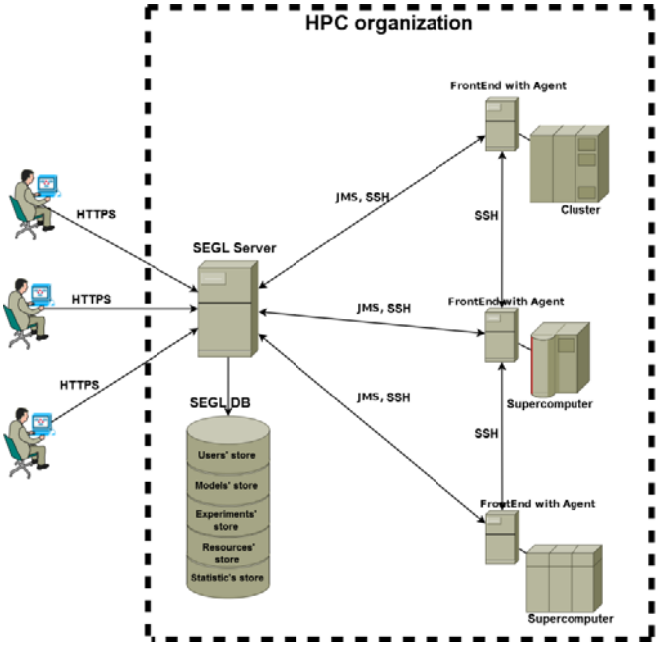

Fig. 3 Architecture of the system

Detail descriptions of the computational resources are kept in a format of GLUE 2[18]. Descriptions of the executable modules and the module implementations are kept in a modified format of JSDL[19].

Selection of the computational resource to run a job depends on the following factors: whether there is an appropriate module implementation, whether the resources are accessible for the user and whether these resources are operable at the moment. It also depends on the current load of the resource and the optimal value of the different benchmarks for the given job[20]. Each resource has a set of the predefined benchmarks values. These values indicate efficiency of the resource for different tasks. And the implementation of the executable module keeps information about the most important benchmarks. This set of the most important benchmarks is ordered in relevance to the computational task. Correlation between resource benchmark values and the list of task benchmarks allows toselect the best resource from the set of available resources.

The system stores all runtime information about the started experiment in the database. The following information is stored: the selected resource for each job, the location of the input and output data on the resource, if the job finished with success or failure and the cause of the failure. This information helps to analyze the experiment later. In addition there is the possibility to continue an experiment which was interrupted.

Jobs are started on the resource by the system agents. The system agent is a service that is started on the front-end machine (Fig. 3). The agent starts and monitors jobs. After job completion the agent analyzes the output data of this job. The agent is controlled by commands of the server. The agent is also responsible for data transfer between resources.

The system has an authorization mechanism, a so called Access Control List (ACL). Users can have different access rights for such system objects as executable modules, experiment models, experiments.

Use cases of the system can be as follows:

- Some user creates an executable module and a set of its implementations. Then this user can grant access to edit and/or use this module.
- Some user creates a model of an experiment with the help of before described modules. Then this user can grant access to edit this model or to create new experiment using this model.
- Some user starts an experiment. Then this user can grant access to monitor this experiment or to get output data.

A graphical client of the system is a desktop application. It is installed, started and updated by Java Web Start framework[21].

User is authenticated in the system by login and password or by X.509 certificate. The system keeps information about allowed resources for the user and about the user accounts on these resources.

Using the client the user describes executable modules, creates experiment models, starts experiments and gets results of the experiments. The client communicates with the server via HTTPS.

V. Executing of the Experiment and Running of the Jobs

Before starting the experiment the user has to specify the location of the input data. The data can be located on the user workstation or on the computational resource. In the last case the user has to have read access to these data. If the data is located on the user workstation, it is transferred from the client to the required resources through the server. Also the result data can be stored at the resources or can be transferred to the user workstation with the help of the client. The first case is more preferred when the data has a large size, for example several terabytes.

The system supports working with parallel file systems, such as Lustre[22]. User can allocate named large disk spaces at the resources, so-called workspaces. The SEGL system allocates only one workspace for each experiment on each resource. As a rule, workspaces have a limited life time. This lifetime can be extended by mechanisms of the parallel file system (limited or unlimited number of extending times, it depends on the file system and its settings). Also extending can be implemented as a system function.

During workflow execution, the server selects a computational resource for each job and then the server transfers the input data of the job to the selected resource. The data is stored on the resource under the account of the user who starts the experiment.

Each user of the system has to add a server public key into a list of the authorized keys (~/.ssh/authorized_keys). After that

the server can connect to some front-end via SSH [3] under the account of the user as it shows Fig. 4. The server transfers the input data to the resource and gets the output data of the job via SFTP protocol.

To submit the job to the queue, to monitor the job state and to analyze the result data of the job the agent is used. The agent is started on the front-end (Fig. 4). To execute some commands, the agent creates a SSH connection from the given front-end to the given front-end under the account of the user. To do it, agent certificate must be added into the list of the authorized certificates. This will be executed by the server automatically.

Message exchange between agent and server is realized by the asynchronous JMS protocol, which enables high productivity and guaranteed message delivery. HTTPS is used as transport layer for JMS.

To start a job, the batch system of the resource [2] is used. The job is submitted to the queue as a script. Then the job will be started under control of the batch system.

The maximum number of jobs in the queue per user is limited. However, many jobs must be started simultaneously in the scope of the experiment. Also each user can start a several experiments simultaneously. To solve this problem the SEGL system has its own internal queues. These queues are part of the workflow engine. There is one internal queue per user and resource in the system. This queue is used for all experiments of the given user on the given resource.

A data transfer from one resource to another is done via SFTP. The agent of the destination resource receives a command from the server. This command contains such information as the source resource, the required data, the user name on the source resource and the user name on the destination resource.

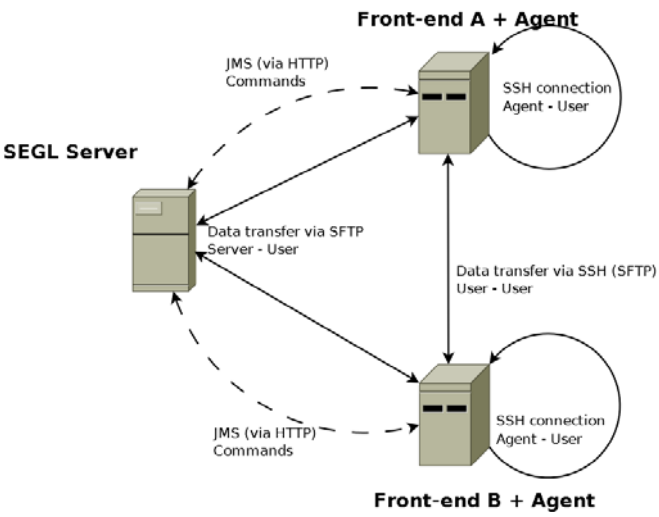

Fig. 4 System communications

The agent of the destination resource creates a local SSH connection under the user's account on the resource. We have selected this approach, because we have some limitations or special conditions which will be described in section VI. Then this agent starts the data transfer from the source resource via SFTP.

At the development stage of the system we have found one more possibility to solve the problem of the resource queue limitation. HPC resources have a lot of nodes and processors. Most jobs use only part of these processors. The user has to specify the number of cores or processors for the job. This number can be defined as a special parameter of the job submitting script. However, the maximum number of jobs per user in the queue is independent on the number of used cores/processors. If you make a common script for a few jobs, you will be able to start multiple real jobs as one from the viewpoint of the queue. In this case the number of real jobs can be larger. If a user starts jobs manually, he or she has to make such scripts manually. In the SEGL system this action can be done automatically. The system is able to generate such scripts merging jobs of different experiments of the same user.

VI. Special Conditions

A significant requirement to the SEGL system is not to require changes to the infrastructure of the HPC organization. The agents which are started on the front-end machines do not have any specific privileges in comparison with other users. This means that a simple user account for the agent will be created identical to the accounts of all other users. This user account does not have root privileges and it does not belong to any privileged groups. It cannot use Linux command *sudo*[23]. Note that using root privileges or *sudo* command often is impossible, because it may affect security of the organization.

The system stores the input data and starts the jobs on the resources under the account of the end user. Therefore all limitations and policies of the organization will be applied to this user. A billing of used resources and disk spaces will be made for this end user too.

Since the system performs all actions on the resources under the account of the end user, it is not necessary to make any changes in the user accounting system of the organization. All security policies, billing principles and user group policies are remaining the same.

To start working with the system you need to have your own user account in the system. All your resource accounts must be added to (described in) the SEGL system. Then the server public key must be added into the list of your authorized keys. That is all. If user wants to stop working with the system, he or she has to remove server certificate from the list of the authorized certificates.

VII. Use Case: Molecular Dynamics Simulation of Proteins

Molecular Dynamics (MD) simulation is one of the principal tools in the theoreticalstudy of biological molecules. This computational method calculatesthe time dependent behavior of a molecular system. MD simulations have provided detailed information on the fluctuations and conformational changes ofproteins and nucleic acids. These

methods are now routinely used to investigatethe structure, dynamics and thermodynamics of biological molecules andtheir interaction with substrates, ligands and inhibitors.

A common task for a computational biologist is to investigate the determinants of substrate specificity of an enzyme. On one hand, the same naturallyoccurring enzyme converts some substrates better than others. One the otherhand, often mutations are found, in nature or by laboratory experiments,which change the substrate specificity, sometimes in a dramatic way.

To understandthese effects, multiple MD simulations are performed consisting of different enzyme-substrate combinations. The ultimate goal is to establish ageneral, generic molecular model that describes the substrate specificity of anenzyme and predicts short- and long-range effects of mutations on structure,dynamics, and biochemical properties of the protein.

While most projects on MD simulation are still managed by hand, largescaleMD simulation studies may involve up to thousands of MD simulations.Each simulation will typically produce a trajectory output file with a size ofseveral gigabytes, so that data storage, management and analysis become aserious challenge. These tasks can no longer be performed by PBS andtherefore have to be automated.

Therefore it is worthwhile to use an experimentmanagement system that provides a language (GriCoL) that is ableto describe all the necessary functionalities to design complex MD parameterstudies. The experiment management system must be combined with the controlof job execution in a distributed computing environment. Fig. 5 showsthe schematic setup of a large-scale MD simulation study. Starting from userprovided structures of the enzyme, enzyme variants (a total of 30) and substrates (a total of 10), in the first step the *Preparation* solver block isused to generate all possible enzyme substrate combinations (a total of 300).

This is accomplished by using the select module in the data flow of the *Preparation*solver block, which builds the Cartesian product of all enzyme variantsand substrates. Afterwards for all combinations the molecular system topologyis built. These topologies describe the system under investigation for theMD simulation program. All 300 topology files are stored in the system database and serve as an input for the *Equilibration* solver block. Forbetter statistical analysis and sampling of the proteins' conformational space,each system must be simulated 10 times, using a different starting conditioneach time. Here the parameterization module of the GriCoL language is used in the data flow of the *Equilibration* solver block to generateautomatically all the necessary input files and job descriptions for the 3000 simulations. The *Equilibration* solver block now starts an equilibrationrun for each of the 3000 systems, which usually needs days to weeks, stronglydepending on the numbers of CPUs available and the size of the system. Inthe equilibration run the system should reach equilibrium. An automatic controlof the system's relaxation into equilibrium would be of great interest tosave calculation time.

This can be achieved by monitoring multiple systemproperties at frequent intervals. The *Equilibrium control* block is usedfor this. Once the conditions for equilibrium are met, the equilibration phasefor this system will be terminated and the *Production* solver block isstarted for this particular system.

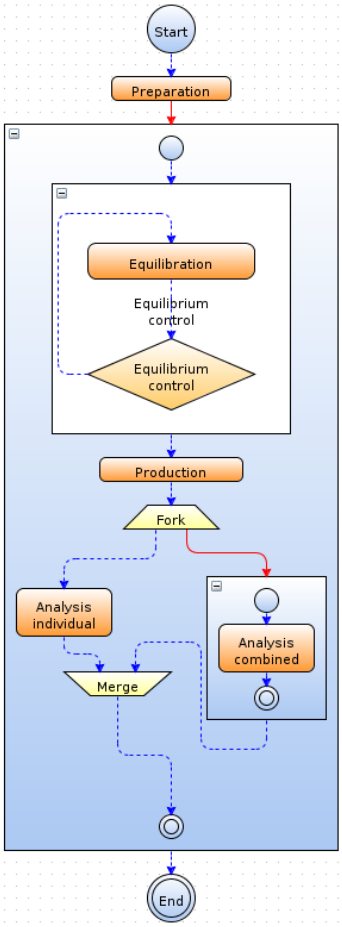

Fig. 5 Bio-molecular experiment

Systems that have not reached the equilibriumyet are subjected to another round of the *Equilibration* solver block. During the *Production* solver block, which performs a MD simulation with a predefined amount of simulation steps, equilibrium propertiesof the system are assembled. Afterwards the trajectories from the production run are subjected to different analysis tools. While some analysis tools arerun for each individual trajectory (*Analysis individual*), some tools need all trajectories for their analysis (*Analysis combined*). The connection line after *Preparation* solver block, as well as connection line before *Analysis combined* block of the control flow is drawn in red-solid,because in these cases all tasks have to be finished before the control flowproceeds to the next block. The blue-dashed lines used to connect the otherblocks indicate that as soon as one of the simulation tasks has finished, itcan be passed to the next block. This example shows the ability of the simple control flow to steer the

laborious process of a large number of single tasks in an intuitive way.

The benefits of using an experiment management systemlike this are obvious. Beside the time saved for setting up, submitting, andmonitoring the thousands of jobs, the base for common errors like misspelling in input files is also minimized. The equilibration control helps to minimizesimulation overhead as simulations which have already reached equilibriumare submitted to the production phase while others that have not are simulatedfurther. The storage of simulation results in the system databaseenables the scientist to later retrieve and compare the results easily.

VIII. Conclusion and Future Work

This paper describes aworkflow system called SEGL that was developed to work in a real HPC organization without any infrastructure changes in such organization.

The SEGL is a high-performance and modular system that provides on the one hand, a convenient way to start complex computational experiments and on the other hand, possibilities to increase quality of HPC services. The implementation of the system is based on the well-known technologies, which are often used in HPC organizations. The use of the system in a Molecular Dynamicssimulation shows the benefits of using the experiment management system.

However, considerable future work remains to be done.System performance in handling large data files and large number of the data sets must be improved. Also we would like to improve the user interface of the client.

The system is installed at the High Performance Computing Center Stuttgart (HLRS) and is used to execute engineering, biological and physical simulation experiments.

In the future we would aim to develop complex planning of the experiment execution. To plan the experiment execution, the system will collect, keep and analyze detail statistical information about all resources of the organization.

References

[1] TOP 500, http://www.top500.org
[2] C. Byun, C. Duncan, andS. Burks, “A Comparison of Job Management Systems in Supporting HPC ClusterTools”, Presentation for *SUPerG*, Vancouver, Canada, 2000.
[3] OpenSSH, http://www.openssh.com/
[4] I. Taylor, E. Deelman, D. Gannon, andM. Shields, *Workflows for e-Science*. Springer Press, 2007.
[5] N. Currle-Linde, F. Boes, P. Adamidis, and M. Resch, “GriCoL: A Language for Scientific Grids”, In *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing (E-SCIENCE '06)*, Amsterdam, Netherlands, 2006.
[6] M. Sonntag, N. Currle-Linde, K. Goerlach, and D. Karastoyanova, “Towards Simulation Workflows With BPEL: Deriving Missing Features from GriCoL”, In *Proceedings of the 21st IASTED International Conference on Modelling and Simulation*,Banff, Alberta, Canada, 2010.
[7] I. Foster, I. Kesselman, andS. Tuecke, “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”,*Int. J. High Perform. Comput.* Appl. 15 3, 2001.
[8] R. Baker, D. Yu, andT. Wlodek, *A Model for Grid User Management, Computing in High Energy and Nuclear Physics.* La Jolla, California, USA, 2003.
[9] T. Antoni, W. Bühler, H. Dres, G. Grein, andM. Roth, “Global grid user support – building a worldwide distributed user support infrastructure”, *Journal of Physics: Conference Series*, 2008.
[10] E. Deelman, D. Gannon, M. Shields, and I. Taylor, “Workflows and e-Science: An overview of workflow system features and capabilities”,*Future Generation Computer Systems*, Volume: 25, Issue: 5 (2009), 528-540.
[11] C. Goble and D. De Roure, “The impact of workflow tools on data-centric research”, In *Data Intensive Computing: The Fourth Paradigm of Scientific Discovery*, T. Hey, S. Tansley, and K. Tolle, Ed. Microsoft Research, 137-145, 2009
[12] S. Callaghan, E. Deelman, D. Guntere, G. Juve, P. Maechling, C. Brooks, K. Vahi, K. Milner, R. Gravesc, E. Field, D. Okaya, and T. Jordan,“Scaling up workflow-based applications”,*Journal of Computer and System Sciences*, 428-446, 2010
[13] V. S. Kumar, P. Sadayappan, G. Mehta, K. Vahi, E. Deelman, V. Ratnakar, J. Kim, Y. Gil, M. W. Hall, T. M. Kurc, and J. H. Saltz,“An integrated framework for performance-based optimization of scientific workflows”, In *Proceedings of the 18th ACM international symposium on High performance distributed computing (HPDC '09)* (Munich, Germany, June 11-13, 2009), 177-186.
[14] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek, “ASKALON: A Grid Application Development and Computing Environment”, In *6th International Workshop on Grid Computing* (New York 2005), 122-131.
[15] M. Wieczorek, R. Prodan, and T. Fahringer,*Scheduling of Scientific Workflows in the ASKALON Grid Environment.*ACM SIGMOD Record, 2005.
[16] Y. Dorozhko, T. Krasikova, Y. Yudin, N. Currle-Linde, andM. Resch, “An Abstract Language and Environment for the Creation and Execution of Experiments over Distributed Computers”, In*Proceedings. of the International Scientific Conference Simulation-2010*, Kiev, Ukraine, 2010.
[17] H. Bouziane, N. Currle-Linde, C. Perez, andM. Resch, “Analysis f Component Model Extensions to support the GriCoL Language”, In *Making grids Work*, pp 45-59, Springer, 2008.
[18] GLUE Specification v. 2.0, http://www.ogf.org/documents/GFD.147.pdf
[19] Job Submission Description Language (JSDL) Specification, Version 1.0, http://www.gridforum.org/documents/GFD.56.pdf
[20] B. Armstrong, H. Bae, R. Eigenmann, F. Saied, M. Sayeed, andY. Zheng, “HPC Benchmarking and Performance valuation with Realistic Applications”, *2006 SPEC Benchmark Workshop (spec)*, 2006.
[21] Java WebStart Overview, http://www.oracle.com/technetwork/java/javase/overview-137531.html
[22] Lustre Cluster FS, http://www.lustre.org/
[23] sudo command, http://www.gratisoft.us/sudo/sudo.man.html