FERMILAB-PUB-08-557-CD

Lattice QCD Workflows

A Case Study

Luciano Piccoli[1,2], James B. Kowalkowski[1], James N. Simone[1], Xian-He Sun[2], Hui Jin[2], Donald J. Holmgren[1], Nirmal Seenu[1], Amitoj G. Singh[1]

[1]Fermi National Accelerator Laboratory, Batavia, IL, USA 60510
[2]Illinois Institute of Technology, Chicago, IL, USA 60616
piccoli@fnal.gov

***Abstract*—This paper discusses the application of existing workflow management systems to a real world science application (LQCD). Typical workflows and execution environment used in production are described. Requirements for the LQCD production system are discussed. The workflow management systems Askalon and Swift were tested by implementing the LQCD workflows and evaluated against the requirements. We report our findings and future work.**

Workflow; workflow management; lattice QCD

I. Introduction

Quantum ChromoDynamics (QCD) is the theory of the strong force that describes the binding of quarks by gluons to make particles such as neutrons and protons. Lattice Quantum ChromoDynamics (LQCD) is the numerical simulation of QCD. Its calculations allow us to understand the results of particle and nuclear physics experiments in terms of QCD. LQCD is both computation and data intensive, and it is representative of large scale scientific computing.

The typical routine of a scientist running a series of LQCD computations can be roughly described by the following steps: copy previous workflow script; change parameters in the file (e.g. quark masses); run the modified script; check periodically whether jobs are running; in case of an error look for the cause in the several produced log files; fix problem and repeat until all the processing succeeds.

As new problems are detected the script is modified to perform additional checking and run recovering procedures. This cycle repeats until achieving a stable version, which is later constantly copied and modified with new application parameters. Despite the ability to produce science, this methodology lacks fundamental aspects that allow scientists to devote most of their time to make science instead of constantly fixing and monitoring workflow scripts.

Groups of scientist within the LQCD collaboration use different tools and methods to coordinate the workflows. Perl, Python or shell scripts are used for invoking the sequence of MPI-based applications; input parameters are kept within the scripts or even hardcoded in the applications themselves; and generated files are stored in a shared area and saved to tape as needed.

Workflow Management Systems (WMS) promote productivity and standardization on this type of environment. WMS are successfully used for scientific applications, such as plasma fusion simulations [1] and earthquake analysis and simulations [2] among others. Our goal is to evaluate existing systems for use in production scale of LQCD experiments.

In this paper we present the LQCD experience with WMS. After an analysis of typical LQCD workflows and environment (section II) we conduct a description of requirements for a WMS (section III). A couple of WMS were tested with LQCD workflows and our findings regarding the requirements are reported on section IV. In section V we discuss the workflow composition and execution. Conclusions and future work are presented on section VI.

II. Lattice QCD Workflows

LQCD computations are typically divided into two workflows: configuration generation and analysis campaigns. The former generates collections of files (ensembles) which are used as inputs to the latter. Both workflows require coordination of physics parameters, cluster parameters, binaries, input and output files. They have distinct flow characteristics as described in the following paragraphs.

The configuration generation workflow (see Fig. 1) is responsible for creating ensembles. An ensemble is an ordered collection of gluon configurations sharing the same physics parameters e.g. lattice spacing (or QCD coupling strength), number of sea quarks and sea quark masses. Configurations are generated in a Markov sequence.

At each step, the last configuration serves as the starting gluon configuration which is evolved forward in simulation time by Monte Carlo techniques. At regular intervals in Monte Carlo simulation time, gluon configuration snapshots are saved. An ensemble may contain a fork where more than one Monte Carlo evolution sequence was started from the same input configuration.

The configuration generation process has two phases: tuning and production. The workflows have a small distinction. The tuning process finishes when self consistency of input parameters and output values is reached, while the production phase repeats for a number of pre-

defined user input steps. In addition to the ensemble, metadata are generated within log and standard output files. Processing of configuration generation accounts for approximately 40% of the total time dedicated for LQCD.

The analysis campaign workflows are a coordinated set of calculations aimed at determining a set of specific physics quantities. For example, predicting the mass and decay constant of a specific particle determined by computing ensemble averaged 2-point functions. A typical campaign consists of taking an ensemble of vacuum gauge configurations and using them to create intermediate data products (e.g. quark propagators) and computing meson n-point functions for every configuration in the ensemble. An important feature of such a campaign is that the intermediate calculations done for each configuration are independent of those done for other configurations.

Fig. 2 shows an analysis campaign for a single configuration file of an ensemble. The complete workflow consists of *n* independent instances of the example in the figure. The *n* outputs form the final workflow output, which is later analyzed. An implicit behavior of analysis campaign is that the number of participants and outputs depend on the input parameters. For example the number of participants generating heavy quark propagators is derived from the amount of certain physics parameters (e.g. number of masses).

LQCD calculations are tightly coupled parallel codes, requiring a high-speed low-latency network interconnection. Therefore configuration generation and analysis campaigns require dedicated hardware. Commodity clusters exclusive for LQCD processing are maintained at national laboratories, using the PBS batch system in conjunction with the MAUI scheduler. The application codes (e.g. MILC and Chroma) are based on the USQCD software suite (available at http://www.usqcd.org/usqcd-software/).

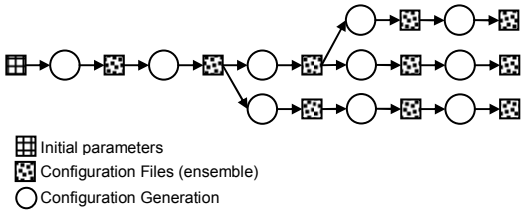

Figure 1. Configuration generation workflow.

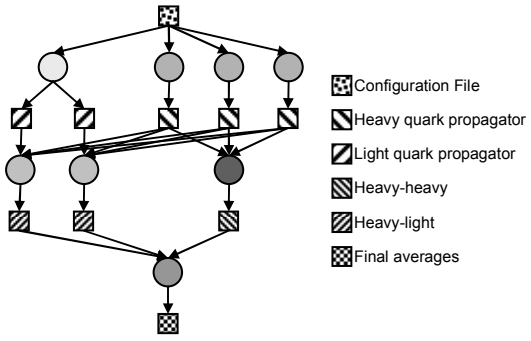

Figure 2. Analysis campaign workflow for one configuration file. Arrows represent the data flow.

III. REQUIREMENTS

In this section we describe the requirements for a WMS based on the LQCD workflows and environment. Some requirements are common for general workflows while others are specific to LQCD and similar scientific applications.

1) Workflow templates: ability to describe generic workflows that can be reuse by changing physics parameters. The use of templates aims at replacing the common procedure of copying workflow script files. Items to consider: description language format, description of participants (workflow tasks), modeling of dependencies, resource constraints and parameterization.

2) Workflow instances: a concrete workflow defined by a template and specific physics parameters. The process of instantiating a workflow is initiated by scientists and therefore should have a simple and straightforward representation. Items to consider: user parameter verification and validation, workflow instance validation, unique instance identifier and instance management control for system administrators.

3) Workflow execution: schedule each workflow task (participant) based on resolved control and data dependencies, by mapping it to available resources (or submitting to PBS). Items to consider: how participants are identified, mapping to resources, dependency resolution and participant scheduling.

4) Progress monitoring: ability to monitor the current status of a workflow instance, allowing quick detection of problems. Complex LQCD workflows may take several weeks to complete and need to be monitored closely by users. Items to consider: track execution, record relevant states and event notification.

5) Workflow execution history: maintain records for accounting and prediction for future workflow executions. Items to consider: execution history and statistics, database support, collection information from monitoring system and execution prediction.

6) Execution of multiple workflow instances: system should support and manage concurrent submission of workflows executions. Items to consider: management and optimization of resource utilization.

7) Quality of service features: certain level of quality of service should be provided: priority of execution based on application and user; expected workflow execution time; limitation of running participants (job throttling) based on the cluster usage. Items to consider: priorities and deadlines, job throttling.

8) Stage in input data files: the ensemble input files for analysis campaigns can be pre-fetched prior to the workflow execution. Items to consider: file pre-fetching and caching.

9) Fault tolerance: provide recovery actions for failed workflow executions. It is acceptable to have a participant as minimum level of fault tolerance, no participant

checkpoint is required. Items to consider: participant and workflow fault tolerance, react to external events.

10) Data provenance: trace files origins, including which workflow instance and participant generated it with which physics parameters. Ability to query parameters and properties of generated files. Items to consider: provenance schema, support for user queries, provenance details.

11) Campaign execution: the long-term running workflows are subject to many types of failures during the execution (e.g. power outage). Items to consider: composite workflows, ability to interrupt workflows and participants and persistent workflow state for extending campaigns.

12) Dispatch campaigns: submission of campaigns (workflow instances) to the system. New campaigns may extend ongoing workflows by adding new participants and dependencies. Items to consider: submission of workflow instances, management of concurrent campaigns and extension of running campaigns (dynamic workflows).

Although all requirements are desirable on a production level WMS, the most important items for the LQCD project in the short term are data provenance, fault tolerance, progress monitoring, campaign execution and execution history. Most requirements reflect the need of a thorough and robust system that can operate in large scale production. The remaining requirements, such as workflow template and instances, arise from the need to quickly compose workflows without burdening users with execution details and allowing focus to be given to the problem resolution. Another great concern not stated directly as a requirement is to have a system capable maximizing aggregate throughput while addressing user quality of service concerns.

IV. Workflow Management Systems

Scientific workflow systems have evolved significantly during the past years, with the development and improvement of systems and workflow languages. Many ideas are borrowed from the business workflow area, although the focus of scientific and business workflows is distinct. Business workflows aim at controlling the flow of processing steps that are usually executed through human interaction, whereas in scientific workflow the importance is given to the data flow, which is passed through the several participants (usually stand alone legacy applications).

The area of scientific workflow systems is a very active research field with several challenging aspects identified [3], some of which are represented by the requirements described in section III.

A variety of WMS targeting scientific workflows are available. Askalon [4], Swift [5], Kepler [6] and Pegasus [7] are a short list. LQCD workflows were tested in Askalon and Swift, while Kepler and Pegasus were considered but not tested.

The typical LQCD workflows were implemented using Askalon and Swift and checked against the requirements. The following sub-sections describe our experience using these two systems on LQCD workflows.

A. Askalon

Askalon is a task coordination and visualization system developed by the DPS research group at the University of Innsbruck. In Askalon, the workflows are specified in Abstract Grid Workflow Language (AGWL) [8], an XML-based language.

Askalon has its own enactment engine to execute workflows according to control flow specifications. Askalon's client interface is independent of the enactment engine and the graphical interface can be used to visualize the progress of executing workflows. Askalon also provides various performance modeling, prediction, instrumentation, measurement, and analysis tools. The system provides a meta-scheduler that distributes the jobs among Grid resources using the Globus middleware services (GRAM). Lower-level scheduling is performed by existing Local Resource Managers (LRMs), such as Condor and PBS.

The following paragraphs describe our findings with regards to the LQCD requirements.

1) Workflow templates: Askalon defines the Abstract Grid Workflow Language (AGWL) to describe workflow models. Resource constraints are currently fixed for the duration of the workflow execution. Constraints are described as RSL parameters and allowed on each activity specification. Input parameters for LQCD workflows are specified in the initial activity of the workflow, and provided as an input to other participants by adding data dependencies from the initial activity. Activities allow binaries, scripts or remote services to be connected through input and output ports. Each activity is unique and registered to the execution engine. Registered activities can be used by multiple workflows.

2) Workflow instances: parameter checking must be done at the workflow level, by adding if-then-else constructs. Instance parameters can be validated at job submission, but verification is limited to Askalon data types. Each workflow instance receives a unique identifier. The management of workflow instantiations not directly supported, however a database does track the current state of all instances.

3) Workflow execution: all activities must be registered before the workflow runs. At run time Askalon invokes the activities according to command line descriptions defined at registration time. Activities are scheduled by the enactment engine through submitting activity jobs through GRAM to the local scheduler after all of the control dependencies of that participant are met.

4) Progress monitoring: the Askalon graphical user interface highlights the activities according to their status, besides displaying a console log. States are also recorded on a Postgres database. There are facilities to register callbacks on predefined events.

5) Workflow execution history: complete execution history tracked by database, providing queue times and wait times. Information from external events can be added to

database. Askalon has support for predicting execution times based on historical data.

6) Execution of multiple workflow instances: the system architecture allows for concurrent workflow management. Independent graphical interfaces are used for submitting and monitoring the execution of workflow instances. Execution is carried out by the execution engine. Optimization of resources can be implemented as the system provides entry points.

7) Quality of service features: workflow and activity priorities can be added by replacing queues with priority queues throughout the Askalon components. Limited support for job throttling is possible by change internal queue sizes.

8) Stage in input data files: no file pre-fetching is available. The workflow must perform any file pre-fetching and caching.

9) Fault tolerance: failed participants are submitted to alternative sites, which is not an option for LQCD workflows. External events from monitoring systems can be added. Currently there is no support for workflow level fault tolerance.

10) Data provenance: the only provenance information provided originates from the monitoring database. It however does not track provenance of data products.

11) Campaign execution: composite activities may include sub-workflows, each one processing an ensemble file. Ability to pause participants is present, but not functional at testing time. The extension of campaigns is currently not possible. There is no tracking of data products required for extending campaigns.

12) Dispatch campaigns: workflow instances can be dispatched to the Askalon system through the provided java web start client interface. Multiple campaigns can be dispatched using different client interface instances. There is no support for controlling groups of workflow. Extension of running campaigns is not possible on the tested version, but support for dynamic workflows is planned for future releases.

Overall the Askalon architecture is flexible and extensible, providing means to add new features according to application requirements. Workflows are intuitive and easy to model through the graphical interface. The monitoring database and visualization interface are great tools for understanding workflow performance and identify bottlenecks. On the other hand the installation process is not trivial on the server side and the graphical interface performs poorly over the network. At this time the system has not been used in actual large scale production.

B. Swift

The Swift Workflow system is an evolution of VDS system [9] being developed at the Computation Institute at the University of Chicago. Workflows are described in the SwiftScript language and executed using the Karajan engine. Karajan uses the Globus toolkit for job scheduling, execution and monitoring.

The following paragraphs describe our findings with regards to the LQCD requirements.

1) Workflow templates: a workflow is described in SwiftScript, which is based on VDL [9]. A graphical editor is not available as in Askalon. Participants are modeled as SwiftScript procedures. Data dependencies between participants are defined by shared input and output parameters to the procedures. The data dependency parameters are file names that must match exactly the names in the file system. Dependencies are resolved based on existence of files. Extensible mappers provide mechanisms to translate file names to actual disk locations. Resource constraints are fixed for duration of the workflow execution, also based on the parameters in RSL specification.

2) Workflow instances: type checking is performed by SwiftScript language. The validity of parameter range must be added by users. Swift provides a command line option (-dryrun) that verifies whether a workflow instance is valid with respect to the language and data types. This option does not guarantee that the workflow will run error free. A unique name and identifier is generated at run time.

3) Workflow execution: at run time the SwiftScript is translated into a Karajan workflow description. The underlying local scheduler is polled for completed tasks. New jobs are launched when dependencies are ready. Participants can only be statically mapped to specific resources via configuration files through unique participant names. Tasks are ready to run as soon as data dependencies are met.

4) Progress monitoring: participant status and errors are logged on a file or standard output for each workflow instance. Event notification is not supported.

5) Workflow execution history: the execution history for each workflow is tracked on unique log files. No database support is provided.

6) Execution of multiple workflow instances: instances of Swift run workflows independently. Management of concurrent workflows and resource utilization are responsibilities of the execution sites.

7) Quality of service features: no quality of service features are provided by Swift. These must be managed at the local scheduler level if possible.

8) Stage in input data files: Swift also does not have any file pre-fetching support. It does however verify whether files are already cached in the system before requesting copies or recreating the files.

9) Fault tolerance: execution of failed participants is retried n times, according to system wide setting. A failed workflow can be resumed by invoking Swift specifying the log file used to track workflow progress. There is no support for reacting to external events.

10) Data provenance: limited support for provenance is provided, although its VDS predecessor has this capability implemented through the virtual data catalog (VDC). Provenance information can be extracted from execution log files produced by Swift and deduced from the SwiftScript itself.

11) Campaign execution: an analysis campaign is described as a SwiftScript; sub-workflows must be contained as functions within the script. Alternatively participants can start independent Swift workflows representing parts of a campaign. There is no support for stopping or pausing participants. Workflow execution state is kept in the log file, allowing restart from a failure. There is no support for extending campaigns.

12) Dispatch campaigns: there is no concept of dispatching a workflow instance to Swift. Workflow instances are executed by directly invoking Swift. Each concurrent workflow instance requires its own instance of Swift and Karajan. Control is individual to each workflow instance. Extension of running campaigns is not possible.

The Swift system is very easy to install and maintain. The precursor system (VDS) has been successfully used on production. Workflow execution in Karajan is lightweight, but Swift is tightly bound to it. The purely data flow procedural syntax is non-intuitive for scientists to learn and program.

V. DISCUSSION

Sample configuration generation and analysis campaign workflows were modeled and tested using both systems. Table I summarizes the functionality of the systems taking in considerations the most relevant requirements described in section III.

TABLE I. SUMMARY OF LQCD REQUIREMENTS AND AVAILABLE WMS FUNCTIONALITY

Requirement	Askalon	Swift
Data provenance	Provenance on execution only	Dependency graph
Participant fault tolerance	Retries on different sites	Retries *n* times
Workflow fault tolerance	None	Resume from failed participant
Progress monitoring	Workflow instance and participant tracking	Status reported to log file
Campaign execution	Available. No campaign extension	Limited support
Execution history	Postgres database	Log files

Among the most relevant requirements the data provenance and fault tolerance for participants and workflows have the highest priority. Data provenance allows scientists to be certain of products origin while fault tolerance contributes for a higher utilization of the dedicated clusters. The experience with workflow composition and execution are described in the following sections.

```
file condition[]<simple_mapper;prefix="cond-",suffix="dat">;
…
iterate i {
  (config[i+1], condition[i+1]) = Tune(config[i])
} until (@extractint(condition[i+1]) > 0);
```

Figure 3. Loop condition expressed as a file dependency in Swift.

A. Workflow Composition

The configuration generation workflow, with no ensemble forking, is quite simple to be modeled. It requires a regular loop construct whose exit condition is based on a simple comparison (e.g. self-consistency check $value[i\text{-}1] - value[i] < error$). Both systems require the creation of additional files to perform the loop condition check, which conforms to their focus on Grid workflows.

In Askalon the process is partially transparent for the scientist composing the workflow through the graphical interface because the mapping to files is performed by the participant implementation. Participants are created and registered before the workflow composition.

On the other hand, the scientist composing the same workflow in Swift must be aware of these extra control files, which contain an integer variable to be tested as the consistency check (e.g. condition is set to 1 the loop is terminated in Fig. 3). This additional requirement illustrates how a workflow definition may be cumbersome to translate into an implementation.

Considering the analysis campaigns, both systems have issues with implementation. In Askalon, control and data dependencies must be modeled. Parallel for loops based on control flow are used to produce the intermediate files based on physics input parameters (e.g. heavy quark propagators). The Askalon implementation of the analysis campaign workflow in Fig. 2 is shown in Fig. 4. The potential parallelism between LQ and HQ is hard to model effectively in Askalon. As a result potential parallelism is lost.

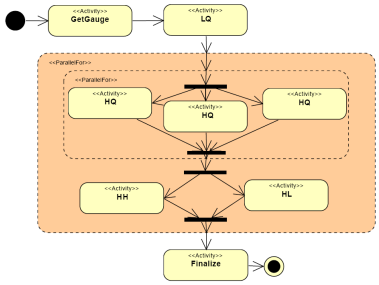

Figure 4. Modified Analysis campaign workflow in Askalon.

```
file hq_propagator[]<tag_array_mapper;
                     properties="input_parameters_file",
                     format="/hq_%wsrc%_%kappaQ%_%gauge%")>;
file lq_propagator[]<tag_array_mapper;
                    properties="input_parameters_file",
                    format="/lq_%wsrc%_m%mass%_nt%tsrc%_%gauge%>;
```

Figure 5. Usage of the customized *tag_array_mapper*. Parameters in file name format are surrounded by the '%' character.

```
heavy_lightfile[i] = HL(lq_propagator[i], hq_propagator[i],
                        hq_propagator[i+4], hq_propagator[i+8]);
```

Figure 6. Association between input files for HeavyLight participant.

In Swift all input and output files generated through the analysis campaign workflow must be explicitly referenced via file names. Collections of files, such as heavy quark propagators, are modeled as an array in Swift. In our implementation, the arrays are constructed via a customized *tag_array_mapper*, which generates the file names based on combinations of workflow input parameters (usage is shown in Fig. 5). The mapping requires users to define all product file names at composition time and construct functions to relate files from various arrays. For example, the generation of a heavy light file requires a light quark propagator file and three heavy quark propagators files according to Fig. 2. For a light quark propagator using ensemble gauge index *i* the heavy quark propagators for the same gauge *i* are needed. However the indices in the array are not contiguous as shown in the sample code in Fig. 6, where heavy quark propagator indices are accessed in a stride pattern.

The composition of configuration generation and analysis campaign workflows in the Askalon graphical model and in the SwiftScript language took several iterations and help of developers. The main reasons were the language learning process and modifications on the workflow structure to fit the constructs and restrictions of each system.

B. *Workflow Execution*

We first successfully installed both systems in our local cluster and configuring them to use the local PBS batch system. The installation of Askalon required more interaction with developer for a proper installation while Swift only needed an extra module to provide access to PBS.

Askalon and Swift ran satisfactorily the workflows and produced expected outputs, although many required features are not available or have limited support as per table I.

VI. CONCLUSIONS AND FUTURE WORK

There is no single system solution for all types of scientific workflows. This paper is an example of how workflow requirements are very dependent on the problem. However it is one of the goals of this project to produce solutions and/or extend existing systems that can be used on similar applications.

Askalon has a well designed and modularized architecture that matches well the LQCD environment and requirements, while Swift is a lightweight system based on a successful data provenance system (VDS/VDC). Both are promising systems, but there are still several issues to pursue before reaching the production level necessary for managing LQCD workflows, especially on critical areas of data provenance and fault tolerance. Developers are working to improve these systems.

The exercise of gathering the requirements helped the LQCD workflow group to bring together computer science and physics aspects of the problem. Understanding well the problem to be solved is the first step on the successful implementation of a scientific workflow. The prototyping of LQCD workflows in different systems allowed us to understand the current issues with WMS and how we can contribute with developers.

Data provenance and fault tolerance are extremely important for LQCD. These are areas in conjunction with campaign execution (workflow cluster scheduling) in which the group is currently working. The Pegasus and Kepler systems are also planned to be evaluated in a similar fashion against the LQCD requirements.

ACKNOWLEDGMENT

This work was supported in part by Fermi National Accelerator Laboratory, operated by Fermi Research Alliance, LLC under contract No. DE-AC02-07CH11359 with the United States Department of Energy (DoE), and by DoE SciDAC program under the contract No. DOE DE-FC02-06 ER41442. The authors would like to thank the Askalon developers Kassian Plankensteiner and Simon Ostermann, and the Swift developers Mihael Hategan, Yong Zhao and Mike Wilde.

REFERENCES

[1] N. Podhorszki, B. Ludaescher and S. A. Klasky. “Workflow automation for processing plasma fusion simulation data”, in Proceedings of the 2nd workshop on Workflows in support of large-scale science, Monterrey, CA. 2007, pp. 35–44.

[2] E. Deelman, et. al. “Managing large-scale workflow execution from resource provisioning to provenance tracking: the CyberShake example”. In Proceedings of the Second IEEE International Conference on E-Science and Grid Computing (December 04–06, 2006), doi:10.1109/E-SCIENCE.2006.99.

[3] Gil, Y. et al. “Examining the challenges of scientific workflows”. Computer, Volume 40, no. 12, Dec. 2007, pp. 24–32.

[4] T. Fahringer, et al. “ASKALON: a grid application development and computing environment”. In Proceedings of the 6th IEEE/ACM international Workshop on Grid Computing (November 13–14, 2005). International Conference on Grid Computing, pp. 122–131 doi:10.1109/GRID.2005.1542733.

[5] Y. Zhao, et al. “Swift: fast, reliable, loosely coupled parallel computation”. In IEEE International Workshop on Scientific Workflows 2007, pp. 199–206.

[6] Altintas, I. “Kepler: an extensible system for design and execution of scientific workflows”. In the 16th Intl. Conference on Scientific and Statistical Database Management (SSDBM), Santorini Island, Greece, June 2004.

[7] Deelman, E. “Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems”. Scientific Programming Journal, Vol. 13(3), 2005, Pages 219–237.

[8] T. Fahringer, S. Pllana, and A. Villazon. “AGWL: abstract grid workflow language”. In International Conference on Computational Science, Programming Paradigms for Grids and Metacomputing Systems, Krakow, Poland, June 2004.

[9] I. Foster, J. Voeckler, M. Wilde and Y. Zhao. “Chimera: a virtual data system for representing, querying, and automating data derivation”. 14th International Conference on Scientific and Statistical Database Management (SSDBM'02).