Data oriented workflow using semantic technologies

Hui Bu*, Ran Liu

School of Software, North China University of Water Resources and Electric Power, China

Received 1 March 2014, www.cmnt.lv

Abstract

Scientific workflows are a topic of great interest in the Grid community that sees in the workflow model an attractive paradigm for programming distributed wide area Grid infrastructures. Scientific workflows have recently emerged as a new paradigm for scientists to formalize and structure complex and distributed scientific processes to enable and accelerate many scientific discoveries. In contrast to business workflows, which are typically control flow oriented, scientific workflows tend to be dataflow oriented, introducing a new set of requirements for system. In this paper, we consider a general workflow setting in which input data sets are processed by a graph of transformations to produce output results. Our goal is to perform efficient selective refresh of elements in the output data, i.e., compute the latest values of specific out-put elements when the input data may have changed. The data provenance is investigated to be used to enable efficient refresh. The proposed approach is based on capturing one level data provenance at each transformation when the workflow is run initially. Then at refresh time provenance is used to determine (transitively) which input elements are responsible for given output elements, and the workflow is rerun only on that portion of the data needed for refresh. The reported preliminary experimental results are developed on the overhead of provenance capture, and on the crossover point be-tween selective refresh and full workflow computation development.

Keywords: scientific workflows, scientific workflow management system, semantic technologies

1 Introduction

Scientific workflows have recently emerged as a new paradigm for scientists to integrate, structure, and orchestrate a wide range of local and remote heterogeneous services and software tools into complex scientific processes to enable and accelerate many scientific discoveries. Significant scientific advances are increasingly achieved through complex sets of computations and data analyses. These computations may com-prise thousands of steps, where each step might integrate diverse models and data sources that different groups develop. The applications and data might be also distributed in the execution environment. The assembly and management of such complex distributed computations present many challenges, and increasingly ambitious scientific inquiry is continuously pushing the limits of current technology.

Workflows have recently emerged as a paradigm for representing and managing complex distributed scientific computations, accelerating the pace of scientific progress. Scientific workflows orchestrate the dataflow across the individual data transformations and analysis steps, as well as the mechanisms to execute them in a distributed environment. Each step in a workflow specifies a process or computation to be executed according to the data flow and dependencies among them. The representation of these computational workflows contains many details required to carry out each analysis step developed within the context of an earthquake science application.

Workflow systems exploit these explicit representations of complex computational processes at various levels of abstraction to manage their life cycle and automate their execution. In addition to automation, workflows can provide the information necessary for scientific reproducibility, result derivation, and result sharing among collaborators. By providing automation and enabling reproducibility, they can accelerate and transform the scientific analysis process.

This traditional best effort execution model commonly does not fully consider the dynamic and course-grain nature of Grid environments dominated by a broad set of performance overheads such as large latencies (several seconds), unpredictable queuing times, sudden unavailability of existing resources, external load on shared-memory computers, unexpected jobs on shared local queues, and inaccurate predictions for new processor architectures, which can severely deteriorate the workflow execution that is likely to deviate from the expected schedules. In addition, although the Grid provides in theory an unlimited amount of compute power, currently existing Grid test beds offer a limited amount of high-performance resources that are important to be used efficiently.

Over the years, performance analysis tools have emerged as an important means for detecting bottlenecks in high performance computing applications for which optimizing compilers rarely deliver satisfactory results. Grid computing is sensitive to similar problems. Running workflows based on pure scheduling techniques without understanding what really happens during the execution

***Corresponding author** e-mail: buhuiemail@126.com

may easily lead to poor execution times and inefficient usage of computing resources. Performance analysis tools are there-fore an important asset for understanding the behaviour and reasons for performance losses required to improve runtime middleware environments and, ultimately, the workflow executions in dynamic Grid infrastructures. Such a tool for high-level performance analysis of scientific Grid work-flows is currently missing.

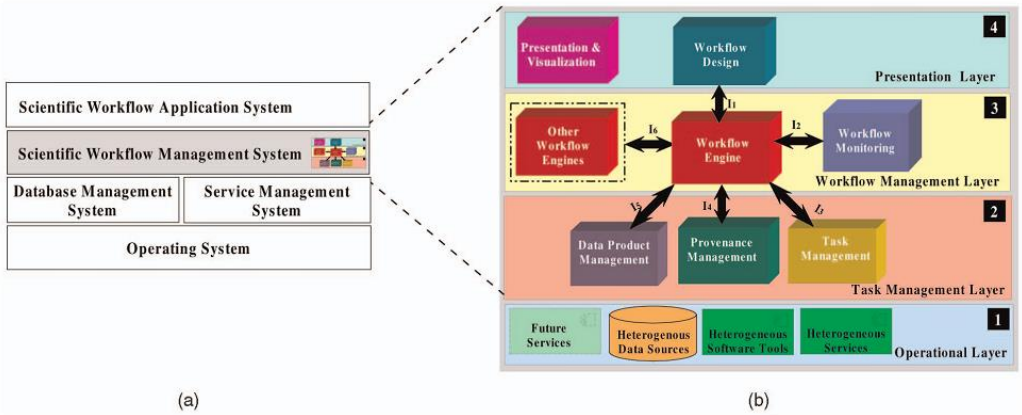

FIGURE 1 a) The position of an SWFMS within a software stack; b) zoom-in view of the reference architecture for SWFMSs

In this paper, we propose a systematic approach to building a tool for performance analysis and steering of scientific workflow applications in heterogeneous and distributed Grid environments. We introduce a theoretical reference parameter, called ideal execution time, that provides a realistic bound for the lowest (that is, "fastest") execution time achievable by a workflow in a certain Grid infrastructure. To address this issue, we propose the first reference architecture for SWFMSs based on a comprehensive survey of the literature and identification of key requirements. According to the proposed reference architecture, we further propose a service-oriented architecture for the VIEW system. Leveraging SOA, VIEW consists of six loosely coupled service components, each of which corresponds to a functional component that is identified in the reference architecture, whose functionality is exposed as a Web service. We implemented the VIEW system to validate the feasibility of the proposed architectures. We present a VIEW based scientific workflow application system (SWFAS), to demonstrate the capabilities of VIEW in support of user interaction intensive, visualization intensive, and compute intensive scientific workflows in a heterogeneous and distributed computing environment.

The rest of the paper is organized as follows. Section 2 identifies the related work of the proposed method. Section 3 proposes our reference architecture for SWFMSs. Section 4 we evaluate five representative SWFMSs using the proposed reference architecture. Finally, Section 5 concludes the paper and comments on future work.

2 Related works

Although the term "scientific workflows" were first coined by Vouk and Singh in 1996 [1] for workflow applications in scientific PSEs, only recently, there is an increasing momentum for the research and development of SWFMSs and their applications, due to the increasingly demanding requirements of many compute-intensive and data intensive scientific applications, enabled by the underlying advances of computing technologies, notably Services computing [2], Grid computing [3], and Cloud computing [4]. Scientific workflows leverage existing techniques developed for business workflows but deviate from them as a result of a different set of requirements raised from a wide range of science and engineering problems [5]. While business workflows are control flow oriented with the mission of carrying out business logic to achieve a business goal, scientific workflows tend to be dataflow oriented and aimed at enabling, facilitating, and speeding up the derivation of scientific results from raw data sets.

There has been a large body of work in lineage and provenance over the past two decades. Surveys are presented in, e.g., [6-8], and formal models for provenance are presented in, e.g., [9-12]. Provenance in the context of schema mappings is studied in [13-15]. None of these papers exploits provenance for selective refresh in a general workflow environment. There also has been a large body of work in incremental view maintenance: the efficient propagation of base data modifications, usually in a relational setting [16, 17]. Our work considers general workflows, rather than relational views. Also, in contrast to the view maintenance problem, selective refresh considers efficiently computing the up-to-date value of individual output elements, rather than keeping the entire view up to date by propagating changes made to the base data. Reference [18] provides a framework to explain "missing" answers in queries. There is some high-level

similarity between how explanations provided by their framework are created and how we support efficient refresh using data provenance. However, the details are quite different, and their framework supports SQL queries rather than general workflows. Reference [19] considers the problem of "update exchange" be-tween data peers linked by mappings. A problem they address is determining when a derived data element is no longer valid, but they do not provide a means to selectively refresh out-of-date values. Also, transformations in [19] are restricted to those that can be expressed in data log.

3 Incorporating semantic to data oriented workflow

Although the reference architecture proposed by WfMC, has been widely used for BWFMSs, this reference architecture does not satisfy key requirements R1-R5 for SWFMSs identified in the previous section. In this section we propose a reference architecture for SWFMSs. As shown in Figure 1b, the reference architecture consists of four logical layers, seven major functional subsystems, and six inter-faces. Figure 1a shows a typical software stack of a scientific workflow application: on top of an operating system, a data management system and a service management are used by an SWFMS for data management and service management, respectively. An SWFAS is developed over an SWFMS by the introduction of additional domain specific application data and functionalities.

3.1 DIFFERENT LAYERS OF THE PROPOSED METHOD

The first layer is the Operational Layer, which consists of a wide range of heterogeneous and distributed data sources, software tools, services, and their operational environments, including high-end computing environments. The separation of the Operational Layer from other layers isolates data sources, software tools, services, and their associated high-end computing environments from the scope of an SWFMS, thus satisfying requirement R5.

The second layer is called the Task Management Layer. Tasks are the building blocks of scientific workflows. Tasks consume input data products and produce output data products. At the same time, provenance is captured automatically to record the derivation history of a data product, including original data sources, intermediate data products, and the steps that are applied to produce the data product. This layer abstracts underlying heterogeneous data into data products, services, and software tools into tasks, and provides efficient management for data products, tasks, and provenance metadata. Therefore, the Task Management Layer satisfies requirements R2, R3, and R4. Moreover, the separation of the Task Management Layer from the Operational Layer promotes the extensibility of the Operational Layer with new services and new high-end computing facilities, and localizes system evolution due to hardware or software advances to the interface between the Operational Layer and the Task Management Layer. The task-level interoperability requirement (R7: level 1) should be addressed in this layer.

The third layer is the Workflow Management Layer, which is responsible for the execution and monitoring of scientific workflows. At this layer, the building blocks of a scientific workflow are the tasks provided by the underlying Task Management Layer. In this layer, an execution of a scientific workflow is called a workflow run, which consists of a coordinated execution of tasks, each of which is called a task run. Therefore, the Workflow Management Layer addresses requirements R6 and R7. The separation of the Workflow Management Layer from the Task Management Layer concerns two aspects as follows:

1) it isolates the choice of a workflow model from the choice of a task model, so changes to the workflow structure do not need to affect the structures of tasks and

2) it separates workflow scheduling from task execution, thus improves the performance and scalability of the whole system. The interoperability of workflows (requirement R7: level 2) has to be addressed by standardizing workflow models, workflow run models, and workflow languages.

The fourth layer is the Presentation Layer, which provides the functionality of workflow design and various user interfaces and visualizations for all assets of the whole system. The Presentation Layer has interfaces to each lower layer (not shown in the figure for simplicity). The separation of the Presentation Layer from other layers provides the flexibility of customizing the user interfaces of the system and promotes the reusability of the rest of system components for different scientific domains. Thus, this separation supports requirement R1. The interoperability of workflows (requirement R7: level 2) should be addressed by standardizing the workflow layout (e.g., look-and-feel) at this layer.

3.2 MIDDLEWARE OVERHEAD

The middleware overhead is due to the work performed by the middleware services to support the proper execution and completion of the workflow, which we further divide into several components based on the service functionality.

Resource brokerage. This represents the time required by the Resource Broker to query the information service and provide to the Scheduler the processors and activity deployments needed to execute the application. Addition-ally, this overhead has an important latency component (few seconds), mostly due to the mutual host authentication. This service latency is a common overhead component present in all our middleware services.

Performance prediction. This represents the time to provide forecast information about the execution time of individual activities on the Grid sites indicated by the Scheduler, for example, using a polynomial fitting heuristic based on historical or training data.

Scheduling. This represents the time to appropriately map the workflow activities onto the Grid resources, which includes the following two sub overheads:

1) a scheduling algorithm, which represents the time required to compute a schedule (often using time-consuming optimization heuristics if the scheduling problem is NP-complete), and

2) rescheduling, which represents the time needed to make a new scheduling decision, for example, because of a performance contract violation or if the workflow changes its runtime structure.

3.3 SUBSYSTEMS

The seven major functional subsystems correspond to the key functionalities required for an SWFMS. Although the reference architecture allows the introduction of additional subsystems and their features in each layer, this paper only focuses on the major subsystems and their essential functionalities.

The Workflow Design subsystem is responsible for the design and modification of scientific workflows. Workflow Design produces workflow specifications represented in a workflow specification language that supports a particular workflow model. One can design and modify a scientific workflow using a standalone or Web-based workflow designer, which supports both graphical- and scripting-based design interfaces. The interoperability of workflows (requirement R7: level 2) should be addressed in this subsystem by the standardization of workflow languages.

The Presentation and Visualization subsystem is very important especially for data-intensive and visualization-intensive scientific workflows, in which the presentation of workflows and visualization of various data products and provenance metadata in multi-dimension is the key to gain insights and knowledge from large amount of data and metadata. These two subsystems are located at the Presentation Layer to meet requirement R1. In this subsystem, the interoperability of workflows (requirement R7: level 2) should be addressed by the standardization of scientific workflow layout.

The Workflow Engine subsystem is at the heart of the whole system and is the subsystem that provides management and execution environments for workflow runs. The Workflow Engine creates and executes workflow runs according to a workflow run model, which defines the state transitions of each scientific workflow and its constituent task runs. The interoperability of workflows (requirement R7: level 2) should be addressed by the standardization of interfaces, workflow models, and workflow run models, so that a scientific workflow or its constituent subworkflows can be scheduled and executed in multiple Workflow.

Engines that are provided by various vendors. In SWFMSs, multiple Workflow Engine subsystems can be distributed, and each Workflow Engine can execute several workflows in parallel.

The Workflow Monitoring subsystem meets requirement R6 and is in charge of monitoring the status of workflow execution during workflow runtime and if failures occur, provides tools for failure handling [18].

The Task Management subsystem addresses heterogeneity and distribution issues (requirement R3) and provides management and execution environment for tasks, according to a task model and task run model, respectively. The interoperability of tasks between various workflow environments (requirement R7: level 1) can be addressed in this subsystem.

The Provenance Management subsystem meets requirement R2 and is mainly responsible for the management of scientific workflow provenance metadata, including their representation, storage, archival, searching, and visualization. The Data Product Management subsystem meets requirement R4 and is mainly responsible for the management of heterogeneous data products. One key challenge for data product management is the heterogeneous and potentially distributed nature of data products, making efficient access and movement of data products an important research problem. The interoperability of data products between various workflow environments (requirement R7: level 1) can be addressed in this subsystem.

3.4 SYSTEM PROTOTYPE

We have built a prototype system that implements all features and algorithms presented in this paper. This system, built initially to support refresh, has been evolving into a more ambitious sys-tem we call Panda (for Provenance and Data), supporting several other aspects of provenance and data in addition to refresh [17].

In this paper we focus our system description on features relevant to refresh. For the time being, all data sets handled by Panda are encoded in relational tables, but as we have seen, our formal under-pinnings and algorithms do not rely on the relational model. The high-level architecture of the Panda system. The main backend is a SQLite server, storing all data sets, relational (SQL) transformations, provenance, and workflow information. The Panda system supports “opaque” transformations programmed in Python; they are stored separately in files.

Users interact with Panda through a simple command-line inter-face; we intend to build a GUI in the near future. There are three types of user commands:

1) Creating or modifying input data sets;

2) Creating transformations that generate newly-defined data sets from existing ones, to build up workflows;

3) Refresh commands.

The Panda Layer processes all user commands: it stores workflow graphs and their transformations, creates and maintains auxiliary provenance tables, generates provenance predicates and forward filters for output elements, and runs the refresh algorithms.

The Panda system also supports transformations specified as SQL queries, including

queries/transformations involving multiple input tables. Provenance predicates are created automatically for SQL transformations, following known definitions and techniques [5, 7, 10]: Single table Select statements are one-one, so their output provenance predicates can select on declared keys from the input data set. Multi-table Select statements generate provenance predicates for each input table separately, again relying on declared keys. Finally, Group-by queries generate provenance predicates based on the grouping attribute(s). The command to create a new SQL transformation is similar to the command shown in Section 8.1, except followed by a SQL query, whose from clause must refer to already-defined tables. The steps performed by the Panda Layer are also similar to those outlined in Section 8.1; forward filters are never needed since SQL queries cannot produce many-many transformations.

When workflows are created and run, Panda stores everything needed to support selective refresh: provenance predicates and intermediate data sets for backward tracing; transformations and forward filters for forward propagation. The Panda system assumes that all transformations, provenance, and workflows satisfy the requirements specified in this paper. Automatically detecting when the requirements are satisfied—particularly the most interesting requirement of workflow safety is an important area of future work. Under the assumption of all requirements being satisfied, Panda supports selective refresh using the exact algorithms given in this paper.

4 Experimental results

We implemented our approach as a distributed online performance analysis tool within the ASKALON programming and computing environment for the Grid. We implemented P-C and P-SC correlators in Python as WSRF-compliant Web services exposing all unprocessed events as reference properties. We translate our formal rule correlation algebra to a lower level open source correlation engine called the Simple Event Correlator.

We employ the PyGnuplot module that is a Python wrapper to the Gnuplot program to display two different kinds of execution graphs in real time, as presented in this section. We describe experiments of applying our tool for online analysis of the main overheads, using the WIEN2k workflow. We chose a problem case that we solved using 193 parallel k-points and a problem size of 8.5, which represents the number of plane waves that is equal to the size of the eigenvalue problem (that is, the size of the matrix to be diagonalized).

We created three groups of peers, one for each city location. The group in Innsbruck consists of four P-Cs and one P-SC, whereas in Linz and Salzburg, we only started one P-SC as we only had one Grid site available. We elected the Innsbruck P-SC as the coordinator.

The first histogram in Figure 2 illustrates a generic technique that we use to represent the major online phases that occur during the workflow execution. We defined for all overheads a performance contract membership function with a low step threshold that generates a contract variable with a critical value at every polling instance (we disabled the rescheduling action). The horizontal line indicates the overhead that holds at any execution instance, whereas the vertical lines are drawn for readability purposes only. The histogram shows that at the beginning of the execution, we experienced some delay due to the operations performed by the middleware services, more precisely the Resource Broker to retrieve the list of available Grid sites and the Scheduler to compute the mapping of the workflow onto the Grid sites. One characteristic of this workflow is that the number of activities in the parallel regions is unknown until the first activity completes its execution. Since this cardinality port is statically unknown, the Scheduler assumed one activity in each case and serialized all workflow activities onto one Grid site. After another preparatory step to create remote directory structures required to run the legacy applications that implement the workflow activities, the Enactment Engine submitted the first activity and therefore added some queuing overhead. After this first activity completed, the output files were broadcasted to all sites, which added a small data transfer overhead.

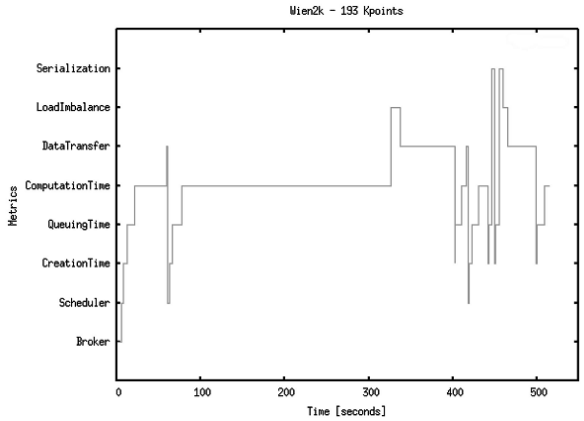

FIGURE 2 Online overhead histogram

After the cardinality port was instantiated, a rescheduling event was triggered upon contract violation, and the Scheduler mapped the new workflow onto the Grid using the Heterogeneous Earliest Finish Time heuristic which added a corresponding overhead. Thereafter, all 193activities were submitted in parallel, since the number of processors available on the Grid test bed was large enough to accommodate them. Since our Grid contains heterogeneous processors, some of the activities completed before the others, which produced a slight load imbalance. As lapw1_k is the most time-consuming activity type of the workflow, we did not experience any significant overhead during this phase, which means that the workflow was performing a useful computation. After this positive execution phase, a large number of files had to be gathered onto one single site where a small activity

called lapp1_fermi processed them, which caused a rapid increase in the severity of the data transfer.

Figure 3 depicts another online-generated histogram that represents the aggregated severity of each analysed over-head at every time instance during execution. The last four sites in our Grid test bed are workstation networks that are automatically restarted in Linux Grid mode during night, weekend, and public holidays and are manually rebooted by students and lecturers during the weekdays in Windows mode for their laboratory classes when they are no longer available for our Grid experiments. After this first positive phase in the workflow execution, we decided to exploit the dynamic characteristic of our Grid test bed and removed the availability of the last four Grid sites by manually turning off the workstations that run the GRAM gatekeepers. This eliminated 51 per cent from the total number of available processors, and as a consequence, the Resource Broker created an appropriate event signalling that at time instance, an important number of computational resources had been lost. The arrival of the (simplified) event and, therefore, the serialization overhead, which indicated that some of the lapw2_k parallel activities had to wait in the queue and were executed sequentially due to the lack of available Grid sites.

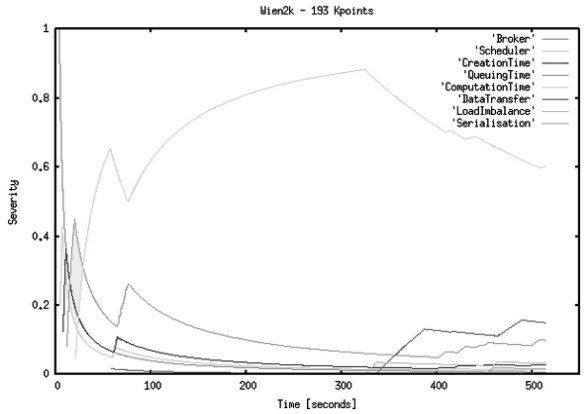

FIGURE 3 Online overhead severity histogram

Our tool is able to automatically generate real-time graphs that represent the chain of events that led to this serialization overhead using the Pydot wrapper to the GraphViz program, as illustrated in Figure 4. At the same time, our tool can automatically produce an online over-head correlation tree for any workflow region at any execution instance. Figure 5 shows this tree for the workflow region indicating that 76.6 per cent of its time is due to the total identified overhead, which is further split into five overheads: queuing, load imbalance, job preparation, data transfer, and serialization, each of them with its own severity value.

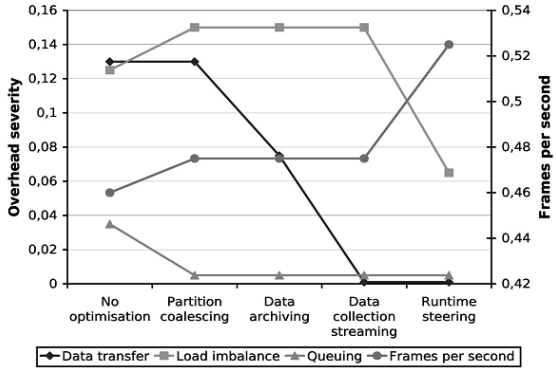

FIGURE 4 Overhead severities for various optimizations.

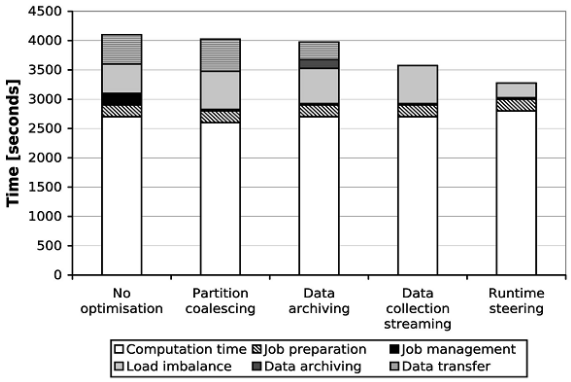

FIGURE 5 Runtime comparison for various optimizations

6 Conclusions

In this paper, we presented a systematic analysis model consisting of a theoretical ideal execution time and a detailed hierarchy of overheads that help the application developers understand the sources of bottlenecks that affect the distributed execution of scientific workflows in heterogeneous Grid infrastructures. We carefully defined the overheads to be as little overlapping as possible, which gives us an important indication of whether any performance loss remained unidentified. We adjusted well-known normalized metrics from parallel processing to the Grid computing scope, including overhead severity, speed-up, and efficiency, which are invaluable parameters to be considered before scheduling when the efficient use of resources is an important issue.

We implemented our approach as a distributed online performance analysis tool within the Grid programming and computing environment. We proposed a distributed super architecture for performance analysis, in which individual peers local to Grid sites correlate large numbers of events to find small sets of meaningful overheads at a higher level of abstraction. To the best of our knowledge, this is the first attempt of applying the event correlation technology, highly successful in the networking field, to the performance analysis of workflow applications in Grid environments. Additionally, we extended the current best effort practice in executing scientific Grid workflows by

defining performance contracts as QoS parameters to be enforced during execution through event-correlation-based fuzzy logic rules. We illustrated experiments for the postmortem and online analysis of two real-world workflow applications with a dynamic structure (that is, statically unknown before the execution) in a real and dynamic Grid environment. The postmortem analysis presents the advantage of detail and rigor through repetitive executions and measurements, whereas the online analysis is invaluable for runtime adaptation, QoS enforcement, and steering. We learned that the serialization of independent activities, load imbalance, job preparation and management, and transfer of large numbers of data dependencies are the most severe overheads for our case-study applications that have to be carefully tuned for achieving good speedup and the efficient use of Grid resources. In this context, we introduced several generic optimization and tuning techniques including workflow partitioning, data archiving, data collection streaming, and runtime steering

References

[1] Berman et al. 2005 New Grid Scheduling and Rescheduling Methods in the GrADS Project *Parallel Programming* **33**(2-3) 209-29
[2] Deelman E et al 2003 Mapping Abstract Complex Workflows onto Grid Environments *Grid Computing* **1**(1) 25-39
[3] Mayer A, McGough S, Furmento N, Lee W, Newhouse S, Darlington J 2003 ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time *Proc. UK e-Science All Hands Meeting* 627-34
[4] Taylor I, Shields M, Wang I, Rana R 2003 Triana Applications within Grid Computing and Peer to Peer Environments *Grid Computing* **1**(2) 199-217
[5] Oinn T, Addis M, Ferris J, Marvin D, Senger M, Greenwood M, Carver T, Glover K, Pocock M, Wipat A, Li P 2004 Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows *Bioinformatics* **20**(17) 3045-54
[6] Erwin D W 2002 UNICORE - a Grid Computing Environment *Concurrency and Computation: Practice and Experience* **14**(13-15) 1395-410
[7] Fahringer T, et al 2007 Askalon: A Development and Grid Computing Environment for Scientific Workflows *Workflows for e-Science: Scientific Workflows for Grids, I J Taylor, E Deelman, D B Gannon, M Shields eds Springer* http://www.askalon.org
[8] Alves A, et al. 2006 Web Services Business Process Execution Language, Specification 2, Organization for the Advancement of Structured Information Standards ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf
[9] Wolski R, Spring N T, Hayes J 1999 The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing *Future Generation Computer Systems* **15**(5-6) 757-68
[10] Czajkowski K, et al 2001 Grid Information Services for Distributed Resource Sharing *Proc 10th IEEE Int'l Symp High Performance Distributed Computing (HPDC)*
[11] D Nurmi, Mandal A, Brevik J, Koelbel C, Wolski R, Kennedy K 2006 Evaluation of a Workflow Scheduler Using Integrated Performance Modelling and Batch Queue Wait Time Prediction *Proc ACM/IEEE Supercomputing Conf (SC)*
[12] DAGMan: Directed Acyclic Graph Manager 2007 *Univ ofWisconsin,Madison* http://www.cs.wisc.edu/condor/dagman/ *Condor Project*
[13] Vaarandi R 2002 SEC—A Lightweight Event Correlation Tool *Proc. Workshop IP Operations and Management (IPOM)*
[14] Liu G, Mok A K, Yang E J 1999 Composite Events for Network Event Correlation *Proc Sixth IFIP/IEEE Int'l Symp Integrated Network Management (IM)*
[15] The Austrian Grid Consortium 2007 http://www.austriangrid.at
[16] PBS: The Portable Batch System 2007 http://www.openpbs.org
[17] Sun Microsystems Sun Grid Engine 2007 http://gridengine.sunsource.net/
[18] Czajkowski K 1998 A Resource Management Architecture for Metacomputing Systems *Proc Fourth Workshop Job Scheduling Strategies for Parallel Processing* 62-82
[19] Foster I, Kesselman C 1997 Globus: A Metacomputing Infrastructure Toolkit *Supercomputer Applications and High Performance Computing* **11**(2) 115-28

Authors

Hui Bu, born in August, 1979, Zhengzhou, Henan Province, China

Current position, grades: master, a lecturer in School of Software, North China University of Water Resources and Electric Power, China.
University studies: computer science and technology.
Scientific interest: computer software, database and software test automation.
Publications: 12 papers.

Ran Liu, born in December, 1979, Zhengzhou, Henan Province, China

Current position, grades: master, a lecturer in School of Software, North China University of Water Resources and Electric Power, China.
University studies: Computer science and technology.
Scientific interest: computer software, database and computer network.
Publications: 12 papers.