Performance Metrics and Ontologies for Grid Workflows

Hong-Linh Truong [a,*] Schahram Dustdar [b] Thomas Fahringer [a]

[a] *Distributed and Parallel Systems Group, Institute of Computer Science, University of Innsbruck*
Technikerstrasse 21A, A-6020 Innsbruck, Austria
{truong,tf}@dps.uibk.ac.at

[b] *Distributed Systems Group, Information Systems Institute, Vienna University of Technology*
Argentinierstrasse 8/184-1, A-1040 Wien, Austria
dustdar@infosys.tuwien.ac.at

Abstract

Many Grid workflow middleware services require knowledge about the performance behavior of Grid applications/services in order to effectively select, compose, and execute workflows in dynamic and complex Grid systems. To provide performance information for building such knowledge, Grid workflow performance tools have to select, measure, and analyze various performance metrics of workflows. However, there is a lack of a comprehensive study of performance metrics which can be used to evaluate the performance of a workflow executed in the Grid. Moreover, given the complexity of both Grid systems and workflows, semantics of essential performance-related concepts and relationships, and associated performance data in Grid workflows should be well described. In this paper, we analyze performance metrics that performance monitoring and analysis tools should provide during the evaluation of the performance of Grid workflows. Performance metrics are associated with multiple levels of abstraction. We introduce an ontology for describing performance data of Grid workflows and illustrate how the ontology can be utilized for monitoring and analyzing the performance of Grid workflows.

Key words: Grid workflows, Grid computing, performance monitoring and analysis, performance metrics and ontology

* Corresponding author. Email: truong@dps.uibk.ac.at

1 Introduction

Recently, Grid workflows have been increasingly exploited as the main programming model for addressing large-scale e-science problems, as demonstrated by a large number of Grid workflow systems [1] and applications [2–4]. As the Grid is diverse, dynamic, and inter-organizational, the execution of Grid workflows is very flexible and complex. Therefore, knowledge about the performance behavior of Grid workflows is required by many Grid middleware services in order to effectively select, compose, and execute workflows in dynamic and complex Grid systems and to tune the workflow performance. Consequently, performance monitoring and analysis tools have to collect, measure, and analyze metrics that characterize the performance of workflows at multiple levels of detail to detect components that contribute to performance problems, and correlations between them.

To understand the performance of Grid workflows, performance metrics of the workflows have to be studied and defined. However, there is a lack of a comprehensive study of useful performance metrics which can be used to evaluate the performance of workflows executed in the Grid. Only few metrics are supported in most existing tools, and most of them being limited at activity (task) level. Moreover, performance data of workflows needs to be shared among various other tools, such as workflow composition, scheduling, and optimization tools. To support a wider dissemination and use of performance knowledge about Grid workflows, essential performance-related concepts and their properties in Grid workflows, together with associated performance data, must be well described. Therefore, an ontology describing performance data of workflows is important because the ontology, like a treaty [5], will facilitate the performance data sharing and can be used to explicitly describe concepts associated with the workflow performance. However, until now, to our best knowledge, such an ontology has not been defined.

In this paper, we present our study on performance metrics of Grid workflows and on the description of performance data of Grid workflows. This paper significantly extends our previous paper [6] by clarifying the hierarchical view of workflows, extending, and refining performance metrics associated with Grid workflows, as well as providing a newly updated version of the ontology used to describe workflow performance information. Our contributions are as follows:

- we introduce a common, hierarchical multiple levels of abstraction model for the performance analysis of Grid workflows.
- we present a large set of performance metrics that associates with relevant concepts within Grid workflows.
- we develop a novel ontology, named **WfPerfOnto**, for describing performance data associated with Grid workflows.

Moreover, we discuss potential applications of the workflow performance metrics and ontologies by illustrating some early work implemented in our tools.

The rest of this paper is organized as follows: Section 2 discusses the workflow and workflow execution model. Section 3 presents performance metrics for workflows. The ontology for describing performance data of workflows is presented in Section 4. We discuss the use of the ontology for performance analysis of Grid workflows in Section 5. Related work is outlined in Section 6. We summarize the paper and give an outlook to the future work in Section 7.

2 Structure and Execution Model of Grid Workflows

A Grid workflow includes a set of dependent activities executed in a Grid environment [7] whose resources are not limited within a single organization. We assume that the real work of activities is performed by operations of services based on WSRF (Web Services Resource Framework) [8], Web services [9], or by executables (e.g., a Java stand-alone program or a C/Fortran application).

2.1 Hierarchical Structure View of a Workflow from a Performance Analysis Perspective

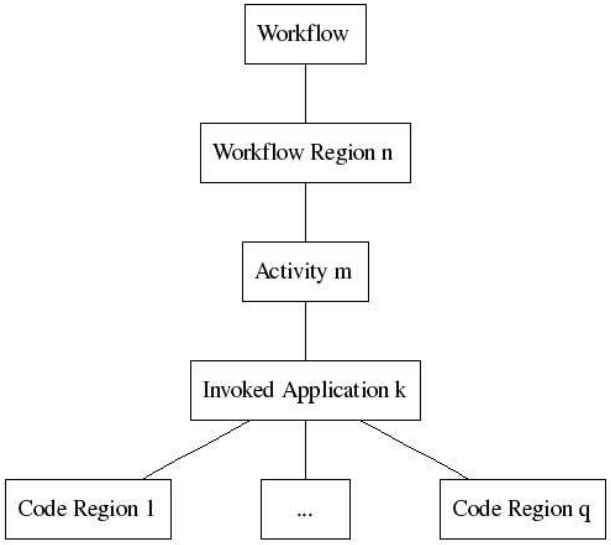

Fig. 1. Hierarchical structure view of a workflow.

From our performance analysis perspective, a Grid workflow (WF) implies a hierarchical model, as presented in Figure 1. At the highest level, we consider the WF as a whole. A WF is considered to consist of WF regions. Our concept of a workflow region is simple: a workflow region consists of a set of workflow

activities that constitutes a single-entry-single-exit region. In this sense, a workflow region can be a sequence of workflow activities, a fork-join, or a do-loop pattern. Workflow regions are analogous to workflow patterns presented in [10,11]. Supporting performance analysis of workflow regions is important as it can provide high level performance information about patterns frequently used in workflows.

An activity [12] represents a task of the workflow. Each activity is associated with one or multiple invoked application(s). However, an activity can also be associated with no invoked application, e.g., an empty or a delay activity (e.g., in BPEL [13]). Furthermore, an activity may process other preparation tasks which are necessary for the execution of the invoked application. For example, in case the invoked application is a WSRF/Web service operation, the activity may deploy the service if the service has not been available at the time the activity starts (e.g., on-demand service deployment [14]) or the activity may find a factory service and ask the factory service to create new service/resource instance. Typically, an activity is associated with one invoked application. Two activities can depend on each other. The dependency between two activities can be data dependency or control dependency.

An invoked application [12] which performs the real work of an activity can be an executable program or a service operation (e.g., of Web services). An executable program is running only when the invoked application starts. If invoked application is a service operation, the service has to be deployed and active before the invoked application can be executed. Invoked applications can be executed in a sequential or parallel manner.

An invoked application is considered as a set of code regions; a code region ranges from a single statement to an entire program unit. A code region can be a local function call, a remote service call, a do-loop construct, an if-then-else construct. Note that the concept of code regions can describe service interactions (e.g., Web Services interaction) as well. For example, if inside service operation *operation*1 of *ServiceA*, *operation*1 invokes operation *operation*2 of service *ServiceB* as follows

```
...
ServiceB.operation2();
...
```

then the call *ServiceB.operation*2() is a code region. To distinguish between different types of code regions, a predefined value can be used to indicate a type of code regions.

Many WF specification languages explicitly provide constructs for specifying workflow regions, such as AGWL [15] and BPEL [13], while others do not have, for example, languages that are based on Petri net [16], and XScufl [17].

However, the hierarchical view of WFs does not prevent us to represent a workflow into the proposed hierarchical structure. For example, we can consider the whole WF as a special WF region or use workflow mining techniques [18–21] to detect workflow patterns and mark these patterns as workflow regions.

2.2 Grid Workflow Execution

A Grid workflow and its components are executed in a Grid infrastructure which includes a set of Grid sites. A *Grid site* is comprised of a set of Grid services within a single organization. A Grid service here should be understood as a computational resource, a middleware service, or a Grid application, based on the OGSA (Open Grid Services Architecture) in which everything in the Grid can be modeled as a Grid service [22]. With respect to computational resources, a Grid site consists of a number of *computational nodes* (or hosts) which are controlled by a single resource management service [1] . A computational node can be any computing platform, e.g, a single-processor workstation, a multi-core computer, an SMP (Symmetric Multi-Processor).

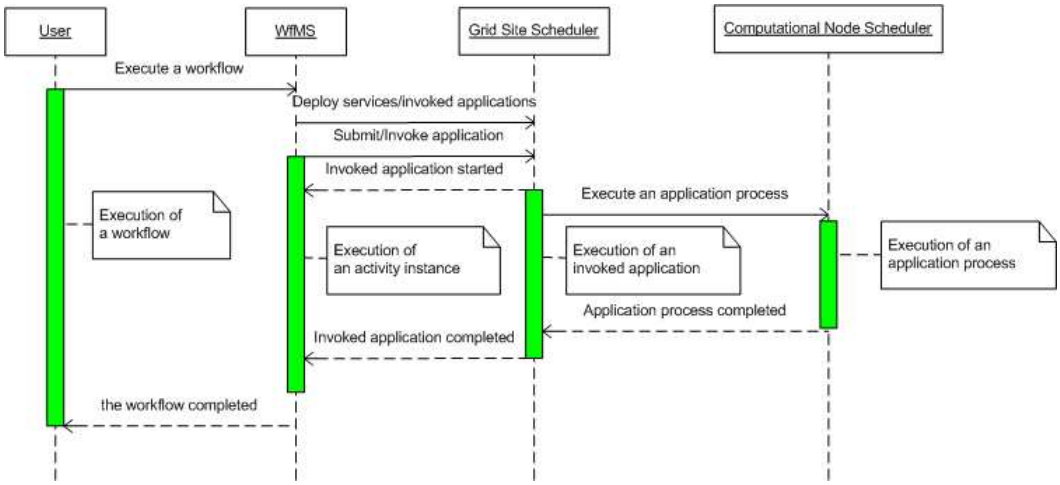

Fig. 2. Simplified execution model of a Grid workflow.

Figure 2 presents the simplified execution sequence of a Grid WF. The user submits a WF to the workflow management system (WfMS). The WfMS instantiates activities. When executing an activity instance, the WfMS locates a Grid site and submits the invoked application of the activity instance to the scheduler of the Grid site. The Grid site scheduler locates computational nodes and executes processes of the invoked application on corresponding nodes.

The execution model presented above is generic enough to cover execution models currently implemented in existing WfMSs. In the Grid, one must assume that there is no central scheduler for the whole Grid which may comprises

[1] Grid site is similar to *IntraGrid* [23,24], *InfraGrid* [24] or *Cluster Grid* [25].

multiple autonomous Grid sites. Moreover, a WfMS has to serve requests from multiple users. Therefore, two layers of scheduling systems, one at the WfMS and the other at Grid sites, exist. In practice, WfMS can schedule activities of workflows and no scheduling is made at Grid sites. For example, in many cases in which invoked applications are Web service operations, whenever an activity is executed, its associated invoked application can be executed without scheduling through an invocation of the Web service operation of a remote Web service on the corresponding computational node. Another situation is that scheduling is conducted at both places. WfMS can schedule an activity in case of a lack of resources or in a multi-user environment. When the invoked application of an activity is submitted to a Grid site, the Grid site scheduler will schedule and locate computational nodes for executing the invoked application. This situation is a typical model for Grid scientific workflows whose invoked applications are executable programs and Grid site schedulers are batch-job ones.

2.3 Activities Execution Model

An invocation of an activity is called an activity instance [12]. An activity instance normally results in an invocation of an invoked application. However, that invocation may fail. In this case, the execution of the activity can be rerun, resulting in another invocation. We, however, consider failed invocations and the successful invocation of the activity under the same activity instance. Each invoked application of an activity instance may be executed on multiple resources, for example, when an invoked application is a parallel program, e.g., an MPI (Message Passing Interface) application.

The execution of workflows/activities is normally modeled by a transition diagram which describes the relationship between workflow/activity events and states, and how the states change [12]. We describe the tracing execution of a workflow/activity using the discrete process model [26]. Let $P(ai)$ be a discrete process capturing the execution of activity instance ai (hence, we call $P(ai)$ the *execution status graph* of an activity instance). A $P(ai)$ is a directed, acyclic graph (N, A), in which N is a set of *nodes* and A is a set of *arcs*. A node is either an *activity execution phase* or an *activity event*; an execution phase basically represents information about a state (transition), e.g., time and name. Let $N = \{E, S\}$ where E is a set of activity events and S is a set of activity execution phases. An arc represents an ordered relation between an exection phase and an event. For every arc $(n_i, n_j) \in A$, then ($n_i \in E$ and $n_j \in S$) or ($n_i \in S$ and $n_j \in E$). With this graph, we can observe, in detail, how the execution of activities changes during runtime. Figure 3 presents an example of a discrete process modeling the execution of an activity instance.

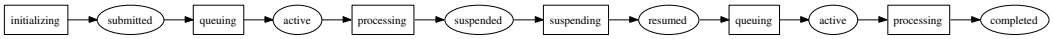

Fig. 3. Discrete process model of the tracing execution of an activity. □ represents an execution phase, ◯ represents an event.

Each execution phase s of an activity instance ai is determined by two events: initial event e_i, and terminal event e_j such that $e_i, e_j \in E$, $s \in S$, and $(e_i, s), (s, e_j) \in A$ of $P(ai)$. To denote an event *name* of $P(ai)$ we use $e_{name}(ai)$; Table 1 presents a few event names which can be used to describe activity events (for possible activity events, see [12]). We use $first(e), next(e), last(e)$ to denote the first occurrence, the consecutive event, and the last occurrence, respectively, of event e in $P(ai)$. We use $t(e)$ to refer to the timestamp of an event e and t_{now} to denote the timestamp at which the analysis is conducted. Because the monitoring and analysis is conducted at runtime, it is possible that an activity instance ai has entered into phase s but there is no such $(s, e) \in A$ of $P(ai)$. When analyzing such a phase s, t_{now} is used as a timestamp to determine the time spent in execution phase s. Note that when using t_{now} in computing performance metrics, the metrics may not be exact values. The correct values are determined based on timestamps of real measurements.

Event Name	**Description**
active	indicate the activity instance has been started to process its work.
completed	indicate the execution of the activity instance has completed.
suspended	indicate the execution of the activity instance is suspended.
failed	indicate the execution of the activity instance has been stopped before its normal completion.
submitted	indicate the activity has been submitted to the scheduling system.

Table 1
Example of event names.

The execution of a Grid workflow involves with multiple Grid sites of different organizations. Thus, time clocks associated with Grid sites may not be synchronized. However, we assume that time clocks of multiple Grid sites are synchronized. Techniques for synchronizing time clocks at different sites are well addressed in literature previously. [2]

3 Performance Metrics of Grid Workflows

Interesting performance metrics of WFs might be associated with many levels of abstraction. We classify performance metrics according to five levels of abstraction, including, from lower to higher level, *code region, invoked application, activity, workflow region* and *workflow*.

[2] see, for example, the bibliography on computer network time synchronization at http://www.eecis.udel.edu/∼mills/biblio.html

In principle, from performance metrics of a lower level, similar metrics can be constructed for the immediate higher level by using appropriate aggregate operators such as sum or average. For example, the communication time spent in one invoked application may be defined as the sum of communication time spent in its code regions. Moreover, there is a question whether metrics should be determined for a specific instance or summarized from multiple instances. When defining metrics, we support both cases. However, exact aggregate methods are dependent on runtime callgraphs, specific metrics and their associated levels. In the following sections we present performance metrics with their associated levels. For a higher level, *we will not show metrics that can be aggregated from that of the lower level*. Instead, we just discuss new metrics which appear at the higher level or an existing metric but it requires a different computing method at different levels of abstraction. We note that the list of metrics is not completed and not all the metrics are useful for analyzing a particular WF. Given a particular WF, only a subset of presented metrics may be of interest.

3.1 Metrics at Code Region Level

Table 2 presents performance metrics of code regions. Performance metrics are categorized into: *execution time, counter, data movement, synchronization, ratio* and *temporal overhead*.

Execution time metrics include total elapsed time (wall-clock time, response time) [3] , user CPU time, system CPU time, CPU time. Counter metrics include performance hardware counters (e.g., L2 cache misses (L2_TCM), number of floating point instructions, etc.) and other counters such as number of calls and of received messages. Performance hardware counters, provided by most contemporary CPU chips, are recently widely used in performance analysis and monitoring, especially for scientific applications [27–29]. Data movement metrics characterize the data transfer such as communication time and exchanged message size. Synchronization metrics describe time spent in the synchronization of executions, such as critical section, condition synchronization, etc. Various ratio metrics can be defined based on execution time and counter metrics such as MFLOPS and cache miss ratio.

If the invoked application is a parallel application (e.g., MPI applications), we can compute *temporal overhead* metrics for code regions. Overhead metrics are based on a classification of temporal overhead for parallel programs [30,31].

[3] Elapsed time, wall-clock time, and response time indicate the latency to complete a task (including IO, waiting time, computation, etc.). These terms are used interchangeably. In this paper, the term *ElapsedTime* refers to elapsed time or response time or wall-clock time.

Category	Metric Name	Description
Execution time	ElapsedTime	Elapsed time of a code region.
	UserCPUTime	CPU time spent in user mode
	SystemCPUTime	CPU time spent in system mode
	CPUTime	Total CPU consumption time
	SerialTime	Time spent in serializing and deserializing data.
	EncodingTime	Time spent in encoding and decoding data.
Counter	L2_TCM, etc.	Hardware counters. The exact number of hardware counters is dependent on specific platforms.
	NumberOfCalls	Number of executions of a code region.
	NumberOfSubs	Number of executions of sub regions within a code region.
	NumberOfSendMsg	Number of messages sent by a code region.
	NumberOfRecvMsg	Number of messages received by a code region.
Data movement	TotalCommTime	Total communication time.
	TotalTransferSize	Size of total data transfered (send and receive).
Synchronization	ExclSynTime	Single-address space exclusive synchronization.
	CondSynTime	Condition synchronization.
Ratio	MeanElapsedTime	Mean elapsed time per execution of a code region.
	CommPerCompTime	Ratio of communication to computation time.
	MeanTransferRate	Mean data transfer rate.
	MeanTransferSize	Mean transfered data size.
	MFLOPS, etc.	Ratio metrics computed based on hardware counters.
Temporal overhead	OCTRP, etc.	This type of metrics is defined only for parallel code regions.

Table 2
Performance metrics at code region level.

Examples of overhead metrics are control of parallelism (denoted by OCTRP), loss of parallelism, etc.

3.2 Metrics at Invoked Application Level

Most performance metrics at code region level can be provided at the invoked application level by using aggregate operators. Table 3 presents extra performance metrics associated with invoked applications.

An invocation of an application can fail due to the failure of underlying systems or applications. Determining whether the failure is due to systems or applications is important, nevertheless, not an easy task. It normally depends on specific errors and the ability of performance monitoring tools.

Computational nodes in the Grid are diverse. An invoked application can be submitted to different nodes and consecutive/parallel invocations of an application are not executed on fixed nodes. Therefore, it is normally difficult, if not impossible, to determine the standard speedup, as in parallel computing or in

Category	Metric Name	Description
Execution time	ElapsedTime	Elapsed time of the invoked application.
	QueuingTime	Time that the local Grid scheduler spends in instantiating application processes.
Counter	NumberOfCalls	Number of executions of the invoked application.
	NumberOfSysFailedCalls	Number of failed invocations due to system problems
	NumberOfAppFailedCalls	Number of failed invocations due to application problems
	NumberOfFailedCalls	Number of failed invocations.
Ratio	FailedCallsRate	Ratio of failure invocations to the total invocations.
Scalability	PerfScaleFactor	Scale factor of the performance between two invocations of the same application.

Table 3
Performance metrics at invoked application level.

homogeneous systems, of an invoked application in various Grid computational nodes. However, based on the performance comparison of historical invocations of the same application, the scheduler, for example, can make a better decision on where to submit the application without knowing the detailed information of computational nodes. For example, during the execution of the workflow, the scheduler can remember the computational nodes which provide better performance and reuse the nodes next time. The $PerfScaleFactor(ia_g, ia_h)$, used to indicate the performance scale factor between two invocations g and h of the same application ia, is defined by

$$PerfScaleFactor(ia_g, ia_h) = \frac{ElapsedTime(ia_g)}{ElapsedTime(ia_h)} \quad (1)$$

where $ElapsedTime(ia)$ is the elapsed time of invoked application ia.

3.3 Metrics at Activity Level

Table 4 presents metrics measured at activity level. Performance metrics can be associated with activities and activity instances. Execution time includes end-to-end response time, processing time, queuing time, suspending time, etc. The processing time of an activity instance ai, $ProcessingTime(ai)$, is defined by

$$ProcessingTime(ai) = \sum_{all}(t(next(e_{active}(ai))) - t(e_{active}(ai))) \quad (2)$$

where $next(e_{active}(ai))$ is not an event indicating a failure or cancellation [4]. If $next(e_{active}(ai))$ has not occurred, it means the execution of ai is currently

[4] Note that the execution time from an active event to a suspended event is considered useful because after a suspended event the activity can resume its work.

Category	Metric Name	Description
Execution time	ElapsedTime	End-to-end response time of an activity instance.
	ProcessingTime	Time an activity instance spends in processing.
	QueuingTime	Time an activity instance spends on queuing system.
	SuspendingTime	Time an activity instance spends on suspending.
	FailureTime	Time an activity takes to do the work but finishes unsuccessful.
	ResSharingTime	Time on which an activity has to share the resource with other activities.
Counter	NumberOfCalls	Number of invocations of an activity.
	NumberOfSysFailedCalls	Number of failed invocations due to the system failure.
	NumberOfAppFailedCalls	Number of failed invocations due to the application failure.
	NumberOfDDFailedCalls	Number of failed invocations due to the data dependency failure.
Data Movement	TotalTransferTime	Total time spent on data transfers.
	InTransferSize	Size of total data transfered to an activity.
	OutTransferSize	Size of total data transfered from an activity to another.
Ratio	ActivityThroughput	Number of successful activity instances over time.
	MeanTimePerInstance	Mean time an activity spent on an instance.
	MeanTransferRate	Data transfer rate between a pair of activities.
Synchronization	SynDelay	Synchronization delay.
	ExecDelay	Execution delay.

Table 4
Performance metrics at activity level.

active, $next(e_{active}(ai)$ can be replaced by t_{now}.

Synchronization metrics for an activity involve with the execution of other activities on which the activity depends. Let $pred(ai)$ be the set of activity instances that must be finished before ai; there is a data dependency or control dependency between ai and any $ai_k \in pred(ai)$. $\forall ai_k \in pred(ai); k = 1, \cdots, n$; synchronization delay and execution delay from ai_k to ai, $SynDelay(ai_k, ai)$ and $ExecDelay(ai_k, ai)$, respectively, are defined by:

$$SynDelay(ai_k, ai) = t(first(e_{submitted}(ai))) - t(e_{completed}(ai_k)) \quad (3)$$

$$ExecDelay(ai_k, ai) = t(first(e_{active}(ai))) - t(e_{completed}(ai_k)) \quad (4)$$

The execution delay comprises of synchronization delay, queuing time, and failure time during queuing phase. If $first(e_{submitted}(ai))$ or $first(e_{active}(ai))$ has not occurred, synchronization or execution delay can be computed based on t_{now}.

Metrics associated with an activity are determined from metrics of activity instances of the activity by using aggregate operators. Aggregated metrics of an activity give summarized information about the performance of the activity that can be used to examine the overall performance of the activity.

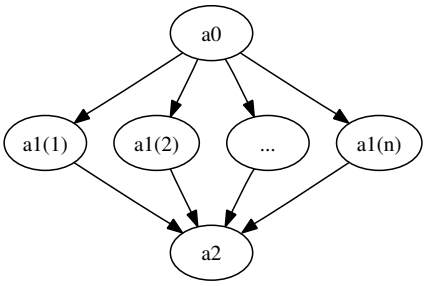

Fig. 4. A fork-join workflow region.

Category	**Metric Name**	**Description**
Execution time	ElapsedTime, ...	Similar to those of activities but for workflow regions.
Counter	NumberOfRedundantActivity	Number of activity instances whose processing results are not utilized. This happens in a *discriminator* construct [11].
Ratio	MeanElapsedTime	Mean elapsed time per invocation of a workflow region.
	PathSelectionRatio	Percent of the selection of a path at a choice region.
Load balancing	ProcessingLoadIm	Load imbalance between activity instances of a fork-join region.
Scalability	PerfScaleFactor	Performance scale factor.
	SlowdownFactor	Slowdown factor for fork-join regions.
Resource	RedundantProcessing	Time spent to process some work but finally the work is not utilized.

Table 5
Performance metrics at workflow region level.

Table 5 presents performance metrics at WF region level. Let SG be a graph of workflow region wr. Let $CP_i =< ai_{i1}, ai_{i2}, \cdots, ai_{in} >$ be a critical path from the initial node to the terminal node of SG. The elapsed time, $ElapsedTime(wr)$, and the processing time, $ProcessingTime(wr)$, of wr are defined as

$$ElapsedTime(wr) = \sum_{k=1}^{n} ElapsedTime(ai_{ik}) \tag{5}$$

$$ProcessingTime(wr) = \sum_{k=1}^{n} ProcessingTime(ai_{ik}) \tag{6}$$

Let wr_g and wr_h be WF regions of a workflow; wr_g and wr_h may be identical region but be executed on different resources at different times. Performance scale factor of wr_g over wr_h, $PerfScaleFactor(wr_g, wr_h)$, is defined by

$$PerfScaleFactor(wr_g, wr_h) = \frac{ProcessingTime(wr_g)}{ProcessingTime(wr_h)} \tag{7}$$

The load imbalance is associated with fork-join WF regions. A simple form of fork-join regions is shown in Figure 4. Load imbalance is defined by

$$ProcessingLoadIm(ai_i) = ProcessingTime(ai_i) - \frac{\sum_{k=1}^{n}(ProcessingTime(ai_k))}{n} \quad (8)$$

SlowdownFactor of a fork-join region is a popular metric for understanding how an imbalance work in parallelizing activities impacts on the performance of the region. It is the inverse of $PerfScaleFactor$ and defined by

$$SlowdownFactor = n \times \frac{max_{k=1}^{n}(ProcessingTime_n(ai_k))}{ProcessingTime_1(ai_1)} \quad (9)$$

where $ProcessingTime_n(ai_k)$ is the processing time of activity ai_k in fork-join region with n activities and $ProcessingTime_1(ai_1)$ is the fastest processing time of activity ai_1 in the (fork-join) region of single activity. Load imbalance and slowdown factor metrics can also be computed for fork-join structures of sub workflow regions. In this case, $ProcessingTime_n(ai_k)$ will be the processing time of a sub-region in a version with n sub-regions.

3.5 Metrics at Workflow Level

Category	**Metric Name**	**Description**
Execution time	ElapsedTime, ...	Similar to those of workflow regions but for workflows.
	ParallelTime	Portion of processing time that workflow activities executed in parallel.
	SequentialTime	Portion of processing time that workflow activities executed in sequential manner.
	ResProcessingTime	Time a resource spends on processing work.
Ratio	QueuingPerElapsedTime	Ratio of queuing time to elapsed time.
	MeanProcessingTime	Mean processing time per activity.
	MeanQueuingTime	Mean queuing time per activity.
	ResUtilization	Ratio of *ResProcessingTime* to the elapsed time of the workflow.
Correlation	ActivityPerRes	Number of activities executed on a resource.
	ResLoadIm	Load imbalance between processing time of resources.
	ActivityDistIm	The imbalance number of activities distributed on computational nodes.
Scalability	PerfScaleFactor	Performance scale factor.

Table 6
Performance metrics at workflow level.

Table 6 presents performance metrics at the workflow level. $PerfScaleFactor$ for a workflow is defined similar to that of workflow regions. Let CP_i be a critical path from the initial node to the terminal node of an execution of workflow wf. The elapsed time, $ElapsedTime(wf)$, and the processing

time, $ProcessingTime(wf)$, of wf are defined based on Equation 5 and 6, respectively. Performance scale factor of workflow wf_g over workflow wf_h, $PerfScaleFactor(wf_g, wf_h)$, is defined by Equation 7. Let $ResProcessingTime(R_i)$ be the processing time performed by computational node R_i [5] . Load imbalance of R_i among computational nodes $\{R_1, \cdots, R_n\}$, $ResLoadIm(R_i)$, is defined by

$$ResLoadIm(R_i) = ResProcessingTime(R_i) - \frac{\sum_{k=1}^{n}(ResProcessingTime(R_k))}{n} \quad (10)$$

The imbalance of the distribution of activities among computational nodes can also be computed as

$$ActivityDistIm(R_i) = ActivityPerRes(R_i) - \frac{\sum_{k=1}^{n}(ActivityPerRes(R_k))}{n} \quad (11)$$

3.6 Performance Metric Ontology

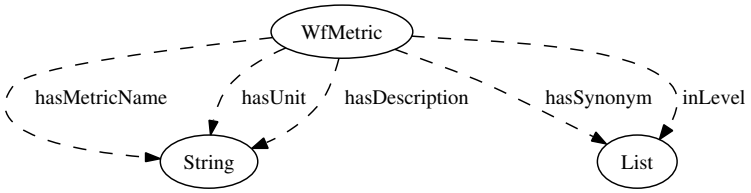

Fig. 5. Description of a WF performance metric.

Proposed performance metrics are described in an ontology named **WfMetricOnto**. A metric is described by class *WfMetric*. Figure 5 presents the concept *WfMetric*. *WfMetric* has five properties. Property *hasMetricName* specifies the name of the performance metric. Property *hasSynonym* specifies other names of the performance metric. Property *hasUnit* specifies the measurement unit of the metric. Property *inLevel* specifies the level with which the metric is associated. Property *hasDescription* explains the performance metric.

3.7 Monitoring and Measurement of Performance Metrics

To provide different metrics at multiple levels of abstraction, performance monitoring and analysis tools for Grid scientific workflows need to operate at multiple levels and to correlate performance metrics from those levels. For analyzing metrics at workflow, workflow region, and activity levels, the tools have

[5] To determine processing time performed by a computational node, we need information of all activities executed on this node. Therefore, this metric is classified into workflow level.

to conduct the monitoring and measurement of WfMSs. Mostly the tools have to collect execution status of workflows and activities from execution engines of WfMS. At this level, monitoring and measurement can be done at centralized or distributed location(s). For analyzing metrics at invoked applications and code regions of invoked applications, the tools have to instrument and measure invoked applications. At this level, the monitoring and measurement are normally conducted at various distributed Grid sites.

The combination of the two different instrumentation mechanisms is a challenging problem. Previously, we have partially addressed some issues on this problem in [32], using static and dynamic instrumentation techniques. Currently, we are developing a comprehensive instrumentation infrastructure that supports the measurement of metrics at all abstraction levels mentioned in this paper [33].

4 Ontology for Performance Data of Workflows

We develop an ontology named **WfPerfOnto** for describing performance data of workflows; **WfPerfOnto** is based on OWL [34]. This section just outlines main classes and properties of **WfPerfOnto** shown in Figure 6.

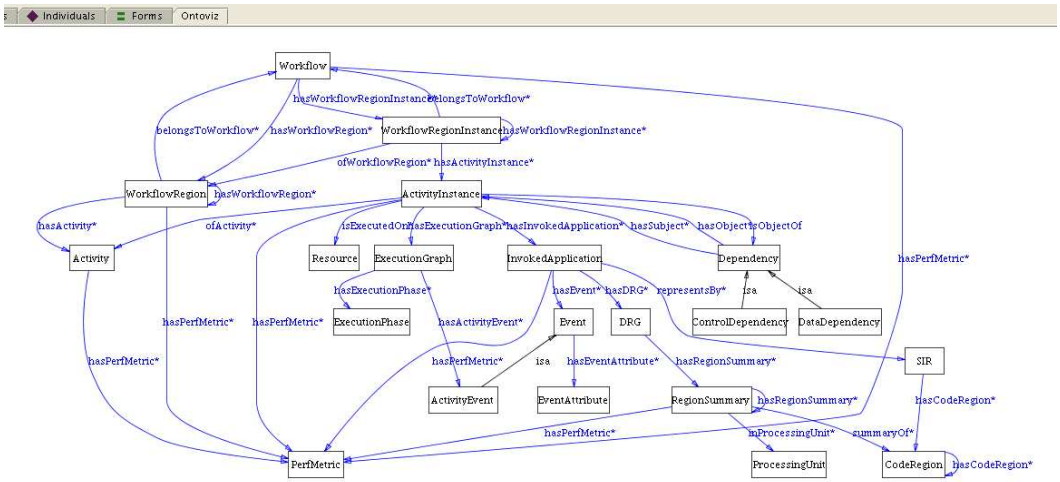

Fig. 6. Part of ontology for describing performance data of workflows visualized within Protege [35].

Workflow describes the workflow (WF). A WF has WF regions (represented by *hasWorkflowRegion* property) and other information. A WF region is described by *WorkflowRegion*. Each WF region has, for example, activities (*hasActivity*), activity instances (*hasActivityInstance*), sub WF regions (*hasWorkflowRegion*).

Activity describes an activity of a WF. *ActivityInstance* describes an activity instance. Each *ActivityInstance*, executed on *Resource*, has an execution graph described by class *ExecutionGraph*. Resource identifies the computational node from which static and dynamic information (e.g., machine name, memory, CPU usage) about computational node can be obtained. Execution graph consists of *ExecutionPhase* and *ActivityEvent* describing activity state and event, respectively. The dependency (control or data) between two activity instances is described by *Dependency*. An *ActivityInstance* is an object or a subject of a dependency; the object depends on the subject. Activity instances have invoked applications (*hasInvokedApplication*).

InvokedApplication describes an invoked application of an activity. Each *InvokedApplication* is associated with a *SIR* (Standardized Intermediate Representation) [36], which represents the structure of the application, including main elements of interest for performance monitoring and analysis, in XML, with a *DRG* (Dynamic Coderegion Callgraph), which represents the dynamic code region call graph [30], and with events occurred inside the application.

The SIR basically contains a set of code regions which are of interest for performance monitoring and analysis. Code region, described by *CodeRegion* includes source code information, such as code region type, source code lines and sub code regions. The dynamic code region call graph, described by *DRG*, consists of region summaries, each stores summary performance measurements of an instrumented code region in a processing unit. A processing unit, described by *ProcessingUnit*, indicates the context in which the code region is executed; the context contains information about the activity identifier, computational node, process identifier and thread identifier. A region summary, described by *RegionSummary* has performance metrics (*hasPerfMetric*) and sub region summaries (*hasChildRS*). *PerfMetric* describes a performance metric, each metric is represented as a tuple of (name, value). The metric name is in **WfMetricOnto**. *Event* describes an event record. Event happens at a time and has event attributes (*hasEventAttr*). *EventAttribute* describes an attribute of an event that has an attribute name and value.

Performance metrics of *Workflow, WorkflowRegion, WorklfowRegionInstance, Activity, Dependency, ActivityInstance, InvokedApplication*, and *RegionSummary* are determined through *hasPerfMetric* property. Each performance metric has name and value; metric names are defined in Section 3.

Note that although **WfPerfOnto** describes relevant performance data of workflows, it does not mean that a tool which implements **WfPerfOnto** has to describe information for all classes and properties in **WfPerfOnto**. Instead, the level of implementation detail should be tool-specific. For example, a tool can describe summary information of activities rather than detailed information of activity instances.

5.1 Common Understanding of Performance Results

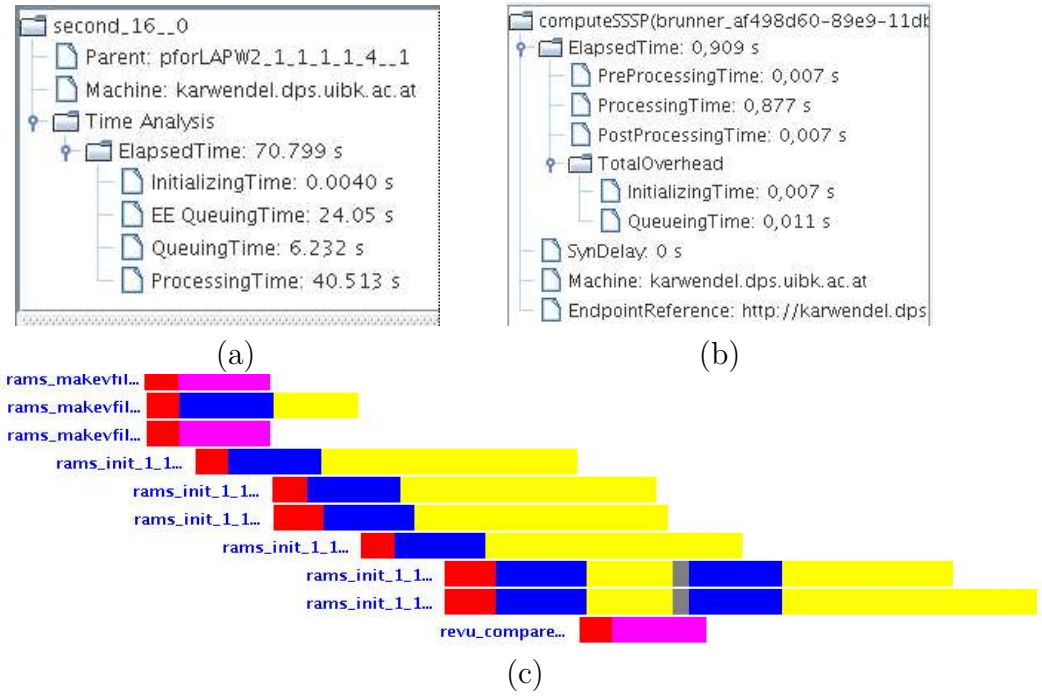

Fig. 7. Examples of workflow performance metrics and execution phases supported: (a) metrics in ASKALON workflow performance tool [37], (b) metrics in K-WfGrid performance tool [38], (c) similar visualization for execution phases in both tools.

In the ASKALON framework [39] and the K-WfGrid project [40], we have developed workflow performance tools that support the proposed performance metrics and concepts. Figure 7 shows simple snapshots of metrics and execution phases implemented in the ASKALON and K-WfGrid performance tools. Figure 7(a) and (b) show metrics associated with activity instance `second_16__0`, provided by ASKALON workflow performance tool [37] and with activity instance `computeSSSP`, provided by the K-WfGrid performance tool [38], respectively. Figure 7(c) shows similar execution phases supported in both tools. The ASKALON toolkit supports a structured workflow language and workflows of executable applications while the K-WfGrid project supports a Petri net workflow language and workflows of Web services. However, both ASKALON and K-WfGrid support common performance metrics and concepts proposed in this paper. Figure 7 gives a simple example of a common view on the performance of workflows from a user's point of view. When different workflow performance tools support a common ontology, the user can benefit from having a common understanding of performance behavior given by these tools for different WfMSs.

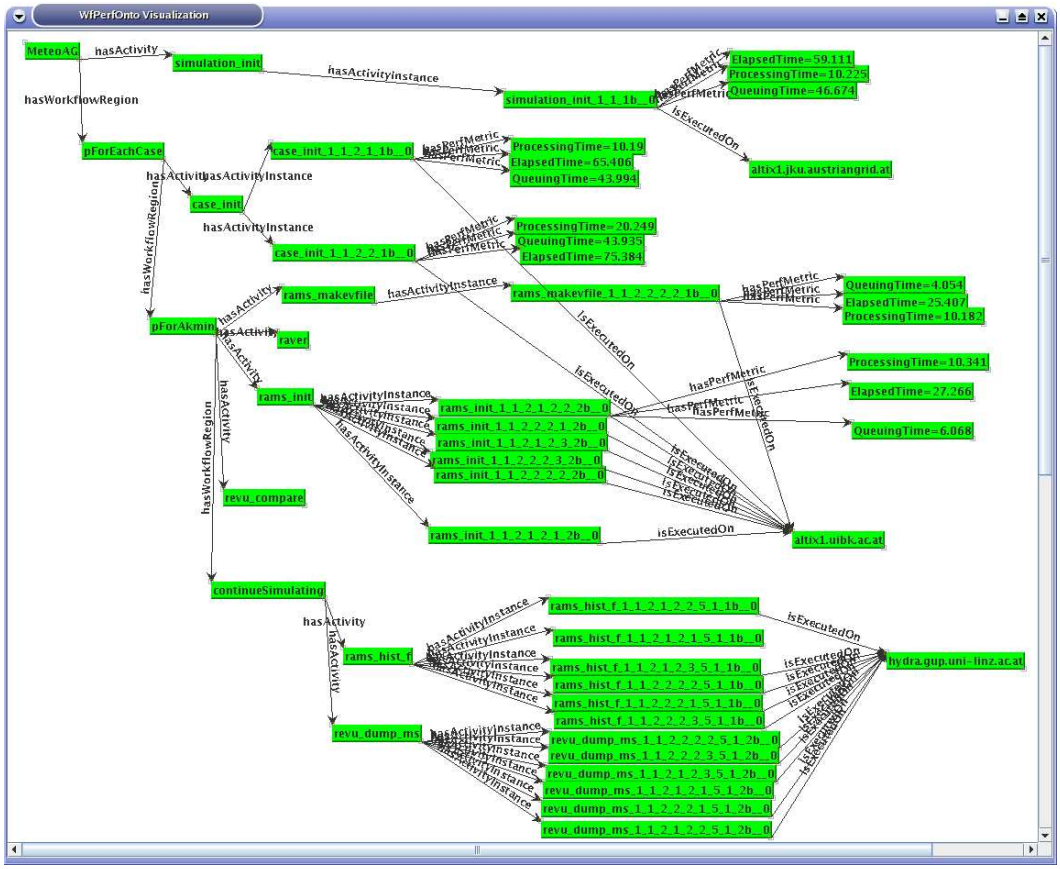

Fig. 8. Part of **WfPerfOnto** for workflow `MeteoAG`.

5.2 Describing and Specifying Performance Data

A performance analysis tool can use workflow performance ontologies to describe performance data of a Grid workflow. Also, ontological performance data can be stored to facilitate the performance knowledge assimilation. For example, when a client of the performance analysis service requests performance results of a workflow, the client can specify the requests based on **WfPerfOnto** (e.g., by using RDQL [41]). The service can use performance ontologies to express performance metrics of the workflow. As performance results are described in a well-defined ontology, the client will easily understand and utilize the performance results.

We are developing tools that can produce ontological performance knowledge, based on **WfPerfOnto**, from online performance monitoring data. Figure 8 presents part of ontological performance data associated with a workflow named `MeteoAG` which is used for meteorological simulations using a numerical limited area model. We can observe in detail the structure of the workflow, including workflow regions (e.g., `pForEachCase, pForAkmin, continueSimulating`), activities (e.g., `simulation_init, case_init, rams_init`), as well as activ-

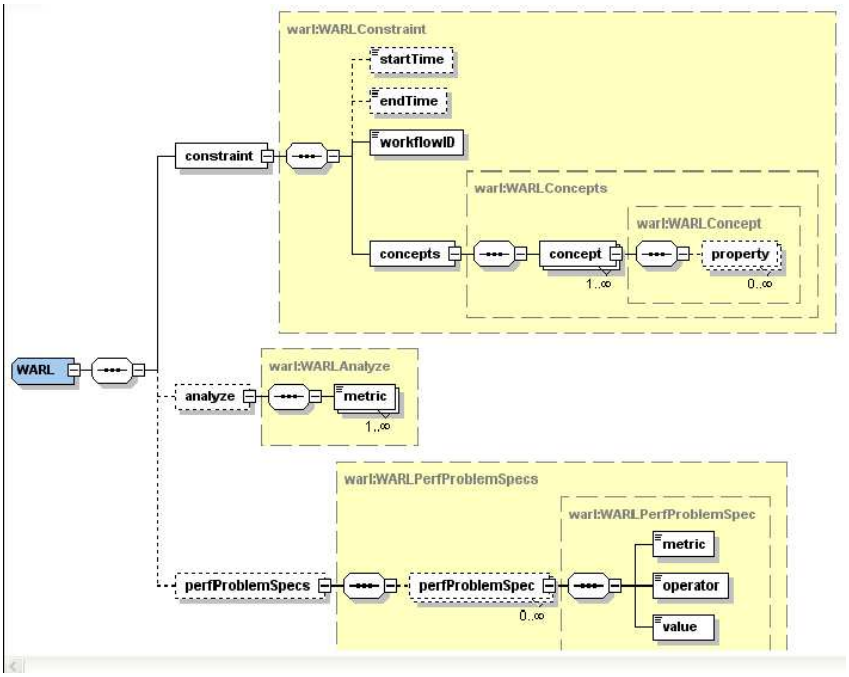

Fig. 9. Workflow analysis request language (WARL).

ity instances, performance metrics, resources. Based on ontological performance data, we could perform further tasks (e.g., reasoning and storing performance data).

Performance metrics, concepts, and properties in **WfPerfOnto**, can be used to specify SLAs (Service Level Agreements) [42,43] for workflows. While an SLA consists of many different information agreed between different partners, one important part of an SLA is a set of performance metrics that specifies constraints on the performance of various components. Well-defined performance-related concepts, properties, and metrics establish standard terms in specifying the expected performance for various components, ranging from a single activity to the whole workflow.

5.3 Workflow Analysis Request Language

Another goal when developing workflow performance ontologies is that well-defined concepts and properties can be used to specify performance analysis requests sent to different performance analysis services which actually analyze events captured during the execution of workflows. Although RDQL can be used to query performance data, it is, however, only suitable for accessing existing ontological performance data, e.g., stored in ontological databases, not for requesting and controlling the task of performance analysis components at runtime.

Our approach is that, during the execution of a workflow, distributed performance analysis services collaborate in fulfilling analysis requests from clients using well-defined concepts and properties defined in workflow performance ontologies, and the ontological performance data is built after the workflow finishes for further analyses. Therefore, we develop an XML-based language which utilizes concepts defined in **WfPerfOnto** and **WfPerfMetric**. Figure 9 presents our first version of a simple workflow analysis request language (WARL), given in [38]. A WARL request includes constraints (type `WARLConstraint`), metrics to be analyzed (type `WARLAnalyze`), performance problems to be checked (type `WARLPerfProblemSpec`). Both constraints and analysis requests are built based on performance metrics and concepts defined in **WfPerfOnto** and **WfPerfMetric**. The following simple request example is used to ask the performance analysis service to analyze two metrics, `ElapsedTime` and `QueueingTime`, for activity named `activity_1` of the workflow identified as `Wien2K_12`.

```
<warl>
<constraint>
    <workflowID>Wien2K_12</workflowID>
    <concepts>
             <concept name="activity_1" type="Activity"/>
    </concepts>
</constraint>
<analyze>
        <metric>ElapsedTime</metric>
        <metric>QueuingTime</metric>
</analyze>
</warl>
```

By supporting the same ontology, different performance analysis services are able to provide performance metrics of diverse applications to different clients which request these metrics by using the same language. This helps simplify the interoperability and integration among performance tools and their clients in the Grid.

6 Related Work

Many techniques have been introduced to study quality of service (QoS) and performance models of workflows, e.g., [44–48]. However, most existing work concentrates on business workflows and Web services processes while our work targets to workflows executed in Grids which are more diverse, dynamic, and inter-organizational. Performance metrics in [44,45] are associated with activity level. Not all the metrics mentioned in this paper are new; various met-

rics have been presented in previous work and existing parallel tools, e.g., in [44,45,32,30,49] but they are targeted to a single (parallel) application or to workflows but at the activity level only. Until now there is no ontology that collects such a large number of metrics. We have defined, collected and associated various metrics with relevant concepts in Grid workflows. Our study considers performance metrics in many levels of detail such as code regions, invoked applications, and workflow regions. Moreover, besides normal performance metrics, e.g., `ProcessingTime` and `QueueingTime`, available in most studies, specific performance metrics, e.g., communication time and temporal overheads, which normally are interesting for scientific workflows/applications are also addressed. Performance metrics are just a subset of QoS metrics. Therefore, many QoS metrics are not considered in this paper, for example, authentication and authorization.

We observed a number of works building QoS metrics and ontologies for Web services. For example, [50] discusses QoS metrics associated with Grid architecture layers. Our work studies performance metrics of Grid workflows. Existing tools supporting performance analysis of workflows, e.g., [51,52], have some performance metrics in common with our metrics. However, our study covers a large set of performance metrics ranging from the workflow level to the code region level. [53] discusses the role of an ontology of QoS metrics for management Web Services. However, there is a lack of such an ontology for Grid workflows. An OWL-based QoS ontology for service-centric systems is presented in [54]. This ontology introduces concepts relevant to QoS such as Time and Dependability. Our ontology includes performance metrics which can be used for specifying QoS. However, performance metrics are not equivalent to QoS metrics. Moreover, our ontology also supports performance metrics of different structured levels of Grid workflows.

Recently, there is a growing effort on mining the workflow [18–21]. Workflow activities are traced and log data is used to discover the workflow model. Events logged, however, are only at activity level. Workflow mining focuses on discovery workflow model from tracing data where our study is to discuss important performance metrics of workflows and methods to describe performance data of workflows. Workflow event logs can be used to analyze performance metrics proposed by our study.

7 Conclusion and Future Work

The performance of Grid workflows must be characterized by well-defined performance metrics. This paper presents a novel study of performance metrics and ontologies for Grid workflows. Performance metrics are associated with multiple levels of abstraction, ranging from a code region to the whole work-

flow. We have introduced a novel ontology that describes essential components of Grid workflows and their relationships and associates those components and relationships to performance data obtained through the performance monitoring and analysis of Grid workflows. We also discussed benefits and illustrated experimental applications of the proposed performance metrics and ontologies for Grid workflows that are currently being implemented in the ASKALON and the K-WfGrid project. In this paper, we present performance metrics that should be collected for Grid workflows, but we do not focus on the instrumentation, monitoring and measuring of these metrics. However, partially, we addressed the measurement of these metrics by introducing a flexible, multi-level instrumentation of Grid scientific workflows [32,33]. We believe that performance metrics we proposed are a useful input for not only sharing performance knowledge of Grid workflows but also composing QoS requests or SLAs.

We have just finished the conceptual part of our approach on using ontology for performance analysis of Grid workflows and started to build our prototype. Still our work presented in this paper is just at an early stage. We need to evaluate and enhance the proposed ontology, and to extend the set of performance metrics. We are working on a prototype of a distributed analysis framework in which performance analysis services use **WfPerfOnto** based requests to exchange analysis tasks when conducting the performance analysis of Grid workflows. In the EU K-WfGrid project [40], we are currently integrating **WfPerfOnto** into the GOM (Grid Organizational Memory) [55], an ontology-based knowledge system for supporting automatic workflow composition and execution. By doing so, various components such as the automatic workflow builder and the workflow optimization tool can reuse existing performance knowledge about workflows executed in the past for composing and optimizing new Grid workflows. How to combine **WfPerfOnto** with existing workflow QoS ontologies is also a future research topic.

Acknowledgments

This paper is a significantly extended version of a paper published in [6]. We thank Peter Brunner for his help in collecting workflow monitoring data. The work described in this paper is partially supported by the European Union through the IST-2002-511385 K-WfGrid project.

References

[1] J. Yu, R. Buyya, A taxonomy of workflow management systems for grid computing, Journal of Grid Computing 3 (3-4) (2005) 171–200.

URL http://dx.doi.org/10.1007/s10723-005-9010-8

[2] E. Deelman, R. Plante, C. Kesselman, G. Singh, M.-H. Su, G. Greene, R. Hanisch, N. Gaffney, A. Volpicelli, J. Annis, V. Sekhri, T. Budavari, M. A. Nieto-Santisteban, W. O'Mullane, D. Bohlender, T. McGlynn, A. H. Rots, O. Pevunova, Grid-based galaxy morphology analysis for the national virtual observatory., in: SC, ACM, 2003, p. 47.

[3] B. Plale, D. Gannon, J. Brotzge, K. Droegemeier, J. F. Kurose, D. McLaughlin, R. Wilhelmson, S. Graves, M. Ramamurthy, R. D. Clark, S. Yalda, D. A. Reed, E. Joseph, V. Chandrasekar, Casa and lead: Adaptive cyberinfrastructure for real-time multiscale weather forecasting., IEEE Computer 39 (11) (2006) 56–64.

[4] P. M. A. Sloot, A. Tirado-Ramos, I. Altintas, M. Bubak, C. A. Boucher, From molecule to man: Decision support in individualized e-health., IEEE Computer 39 (11) (2006) 40–46.

[5] Interview Tom Gruber, AIS SIGSEMIS Bulletin 1(3), http://www.sigsemis.org/ (October 2004).

[6] H. L. Truong, T. Fahringer, F. Nerieri, S. Dustdar, Performance metrics and ontology for describing performance data of grid workflows., in: CCGRID, IEEE Computer Society, 2005, pp. 301–308.

[7] I. Foster, C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[8] OASIS Web Services Resource Framework (WSRF) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[9] W3C: Web Services Architecture, http://www.w3.org/tr/ws-arch/.

[10] W. M. P. V. D. Aalst, A. H. M. T. Hofstede, B. Kiepuszewski, A. P. Barros, Workflow patterns, Distrib. Parallel Databases 14 (1) (2003) 5–51.

[11] Workflow Patterns, http://is.tm.tue.nl/research/patterns/patterns.htm.

[12] Worldflow Management Coalition: Terminology and glossary. technical report wfmc-tc-1011, feb 1999.

[13] Business Process Execution Language for Web Services, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[14] E.-K. Byun, J.-W. Jang, W. Jung, J.-S. Kim, A dynamic grid services deployment mechanism for on-demand resource provisioning, in: CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2, IEEE Computer Society, Washington, DC, USA, 2005, pp. 863–870.

[15] T. Fahringer, J. Qin, S. Hainzer, Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language, in: Proceedings of IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005), IEEE Computer Society Press, Cardiff, UK, 2005.

[16] M. Alt, A. Hoheisel, H. W. Pohl, S. Gorlatch, A grid workflow language using high-level petri nets., in: R. Wyrzykowski, J. Dongarra, N. Meyer, J. Wasniewski (Eds.), PPAM, Vol. 3911 of Lecture Notes in Computer Science, Springer, 2005, pp. 715–722.

[17] XScufl Language Reference, http://www.ebi.ac.uk/t̃mo/mygrid/XScuflSpecification.html.

[18] W. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1128–1142.

[19] J. Herbst, D. Karagiannis, Workflow mining with inwolve, Comput. Ind. 53 (3) (2004) 245–264.

[20] W. Gaaloul, S. Bhiri, C. Godart, Discovering workflow transactional behavior from event-based log., in: CoopIS/DOA/ODBASE (1), 2004, pp. 3–18.

[21] S. Dustdar, T. Hoffmann, W. van der Aalst, Mining of ad-hoc Business Processes with TeamLog, Data and Knowledge Engineering.

[22] Ian Foster et. al, The Open Grid Services Architecture, Version 1.0, global grid forum (January 2005).

[23] I. Redbooks, Introduction to Grid Computing with Globus, IBM, 2003.

[24] J. Joseph, M. Ernest, C. Fellenstein, Evolution of grid computing architecture and grid adoption models., IBM Systems Journal 43 (4) (2004) 624–645.

[25] The Three Types of Grids, http://nz.sun.com/2002-0708/grid/types.html.

[26] J. F. Sowa, Knowledge Representation: logical, philosophical, and compuational foundations, Brooks/Cole, Pacific Grove, CA, 2000.

[27] M. M. Tikir, J. K. Hollingsworth, Using hardware counters to automatically improve memory performance, in: SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, IEEE Computer Society, Washington, DC, USA, 2004, p. 46.

[28] S. Browne, J. Dongarra, N. Garner, K. S. London, P. Mucci, A scalable cross-platform infrastructure for application performance tuning using hardware counters., in: SC, 2000.

[29] W. Mathur, J. Cook, Improved estimation for software multiplexing of performance counters., in: MASCOTS, IEEE Computer Society, 2005, pp. 23–34.

[30] H.-L. Truong, T. Fahringer, SCALEA: A Performance Analysis Tool for Parallel Programs, Concurrency and Computation: Practice and Experience 15 (11-12) (2003) 1001–1025.

[31] J. Bull, A Hierarchical classification of Overheads in Parallel Programs, in: P. C. I. Jelly, I. Gorton (Ed.), Proceedings of Firs IFIP TC10 International Workshop on Software Engineering for Parallel and Distributed Systems, Chapman Hall, 1996, pp. 208–219.

[32] H.-L. Truong, T. Fahringer, S. Dustdar, Dynamic Instrumentation, Performance Monitoring and Analysis of Grid Scientific Workflows, Journal of Grid Computing 3 (1-2) (2005) 1–18.

[33] B. Balis, H.-L. Truong, M. Bubak, T. Fahringer, K. Guzy, K. Rozkwitalski, An Instrumentation Infrastructure for Grid Workflow Applications, 2006, on submission.

[34] OWL Web Ontology Language Reference, http://www.w3.org/tr/owl-ref/.

[35] Protege, http://protege.stanford.edu/.

[36] C. Seragiotto, H.-L. Truong, T. Fahringer, B. Mohr, M. Gerndt, T. Li, Standardized Intermediate Representation for Fortran, Java, C and C++ Programs, Tech. rep., Institute for Software Science, University of Vienna (October 2004).

[37] P. Brunner, H. L. Truong, T. Fahringer, Performance monitoring and visualization of grid scientific workflows in askalon., in: M. Gerndt, D. Kranzlmüller (Eds.), HPCC, Vol. 4208 of Lecture Notes in Computer Science, Springer, 2006, pp. 170–179.

[38] H.-L. Truong, P. Brunner, T. Fahringer, F. Nerieri, R. Samborski, B. Balis, M. Bubak, K. Rozkwitalski, K-WfGrid Distributed Monitoring and Performance Analysis Services for Workflows in the Grid, in: 2nd IEEE International Conference on e-Science and Grid Computing, IEEE Computer Society, Amsterdam, The Netherlands, 2006.

[39] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, M. Wieczorek, ASKALON: A Grid Application Development and Computing Environment, in: 6th International Workshop on Grid Computing (Grid 2005), IEEE Computer Society Press, Seattle, USA, 2005.

[40] K-WF Grid Project. http://www.kwfgrid.net.

[41] RDQL: RDF Data Query Language, http://www.hpl.hp.com/semweb/rdql.htm.

[42] A. Leff, J. T. Rayfield, D. M. Dias, Service-level agreements and commercial grids, IEEE Internet Computing 07 (4) (2003) 44–50.

[43] C. K. Fung, P. C. K. Hung, R. C. Linger, G. H. Walton, Extending business process execution language for web services with service level agreements expressed in computational quality attributes, in: HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 7, IEEE Computer Society, Washington, DC, USA, 2005, p. 166.1.

[44] K.-H. Kim, C. A. Ellis, Performance analytic models and analyses for workflow architectures, Information Systems Frontiers 3 (3) (2001) 339–355.

[45] J. Cardoso, A. P. Sheth, J. A. Miller, Workflow quality of service., in: K. Kosanke, R. Jochem, J. G. Nell, A. O. Bas (Eds.), ICEIMT, Vol. 236 of IFIP Conference Proceedings, Kluwer, 2002, pp. 303–311.

[46] L. jie Jin, F. Casati, M. Sayal, M.-C. Shan, Load balancing in distributed workflow management system, in: Proceedings of the 2001 ACM symposium on Applied computing, ACM Press, 2001, pp. 522–530.

[47] M. C. Jaeger, G. Rojec-Goldmann, G. Mühl, QoS Aggregation for Service Composition using Workflow Patterns, in: Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC 2004), IEEE CS Press, Monterey, California, USA, 2004, pp. 149–159.

[48] F. Rosenberg, C. Platzer, S. Dustdar, Bootstrapping performance and dependability attributes ofweb services, icws 0 (2006) 205–212.

[49] T. Fahringer, M. Gerndt, B. Mohr, F. Wolf, G. Riley, J. Träff, Knowledge Specification for Automatic Performance Analysis, Tech. rep., APART Working group (August 2001).

[50] D. A. Menasce, E. Casalicchio, Quality of Service Aspects and Metrics in Grid Computing, in: Proc. 2004 Computer Measurement Group Conference, 2004.

[51] B. T. R. Savarimuthu, M. Purvis, M. Fleurke, Monitoring and controlling of a multi-agent based workflow system, in: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation, Australian Computer Society, Inc., 2004, pp. 127–132.

[52] A. F. Abate, A. Esposito, N. Grieco, G. Nota, Workflow performance evaluation through wpql, in: Proceedings of the 14th international conference on Software engineering and knowledge engineering, ACM Press, 2002, pp. 489–495.

[53] V. Tosic, B. Esfandiari, B. Pagurek, K. Patel, On requirements for ontologies in management of web services, in: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web, Springer-Verlag, 2002, pp. 237–247.

[54] G. Dobson, R. Lock, I. Sommerville, Qosont: a qos ontology for service-centric systems., in: EUROMICRO-SEAA, IEEE Computer Society, 2005, pp. 80–87.

[55] B. Kryza, R. Slota, M. Majewska, J. Pieczykolan, J. Kitowski, Grid organizational memory: provision of a high-level grid abstraction layer supported by ontology alignment., Future Generation Comp. Syst. 23 (3) (2007) 348–358.