Scientific Programming 16 (2008) 205–216
DOI 10.3233/SPR-2008-0257
IOS Press

Monitoring of Grid scientific workflows

Bartosz Balis [a,*], Marian Bubak [b] and Bartłomiej Łabno [c]
[a] *Institute of Computer Science, AGH University of Science and Technology, Krakow, Poland*
[b] *Institute of Computer Science & ACC CYFRONET AGH, Krakow, Poland*
[c] *ACC CYFRONET AGH, Krakow, Poland*

Abstract. Scientific workflows are a means of conducting in silico experiments in modern computing infrastructures for e-Science, often built on top of Grids. Monitoring of Grid scientific workflows is essential not only for performance analysis but also to collect provenance data and gather feedback useful in future decisions, e.g., related to optimization of resource usage. In this paper, basic problems related to monitoring of Grid scientific workflows are discussed. Being highly distributed, loosely coupled in space and time, heterogeneous, and heavily using legacy codes, workflows are exceptionally challenging from the monitoring point of view. We propose a Grid monitoring architecture for scientific workflows. Monitoring data correlation problem is described and an algorithm for on-line distributed collection of monitoring data is proposed. We demonstrate a prototype implementation of the proposed workflow monitoring architecture, the GEMINI monitoring system, and its use for monitoring of a real-life scientific workflow.

Keywords: Monitoring, instrumentation, Grid, scientific workflows

1. Introduction

Scientific workflows are used by the scientists to perform *in silico* experiments. They provide numerous benefits, among others, the automation of a computation process, ease of use important for non-IT experts, and opportunity to share and reuse experience in the scientific community. Monitoring of scientific workflows is important in many scenarios, to name a few typical ones:

- An application developer is using a tracing and profiling information concerning a workflow run in order to find performance bottlenecks.
- A scientist is searching for past experiments executed as workflows which satisfy specific characteristics in order to compare his/her results to those obtained by other scientists.
- A scientist is searching for a provenance information concerning a data item in order to learn what process (workflow) was used to produce this data.
- A scientist is using a recorded execution of a workflow in order to repeat an experiment, perhaps slightly changing its conditions.
- A scheduler is looking for information concerning invocations of specific services, gathered from multiple workflow runs, in order to optimize the usage of computational resources for a next run.

Until recently, scientific computations were performed mainly via tightly coupled, homogeneous, parallel applications running on clusters. However, scientific workflows are better characterized as loosely-coupled, heterogeneous distributed applications, often running on Grids. Consequently, existing tools for monitoring of applications are not suitable for scientific workflows which pose new challenges for monitoring.

In this paper, key challenges related to monitoring of Grid scientific workflows are presented. We specifically focus on the design of a Grid workflow monitoring architecture, and the problems of distributed on-line collection and correlation of workflow monitoring data in a Grid. Section 2 presents the related work. Section 3 describes the key challenges in monitoring of Grid scientific workflows. Section 4 presents the design of a Grid workflow monitoring infrastructure, and its prototype implementation, the GEMINI monitoring system. Section 5 focuses on the support for on-line collection of workflow monitoring events. Section 6 describes the problem of workflow monitoring data correlation. A monitoring example of a Coordinated Traffic Management (CTM) workflow is presented in Section 7.

*Corresponding author: E-mail: balis@aqh.edu.pl.

2. Related work

There are few systems that address monitoring of Grid scientific workflows. PPerfGrid [14] deals with collection and storage of large amounts of heterogeneous performance data of scientific parallel applications. Mercury [17] supports semi-online monitoring of parallel MPI applications in the Grid. The J-OCM monitoring system and the JMT performance measurement tool [11] support on-line performance measurement of Java applications using custom user-defined metrics.

However, none of the mentioned tools address the instrumentation and monitoring of Grid scientific workflows. None of those tools features a runtime instrumentation service supporting heterogeneous applications, developed in multiple programming languages.

Based on the analysis of the requirements posed by Grid scientific workflows and a review of existing work, some aspects of our solution have been adopted from existing ones, others have been genuinely solved within this work. The monitoring and instrumentation protocols used in our work have been conceived in SCALEA-G [22] and later developed in the collaboration of the authors of this work and the authors of the SCALEA-G. The results of the collaboration have been published in a number of papers [2,6,20,21]. The goal of the collaboration was to create a performance monitoring and analysis infrastructure for Grid scientific workflows. SCALEA-G focuses mainly on the performance analysis of workflows, and, compared to our approach, it lacks, among others, the support for monitoring of legacy codes in a unified framework, a systematic approach to the correlation of the monitoring data, and a support for scalable on-line collection of monitoring events.

The concept of a Standardized Intermediate Representation (SIR) [18] was also adopted from a previous work within the APART working group[1] and developed into SIRWF (Standard Intermediate Representation for WorkFlows) within the mentioned collaboration.

For monitoring of Grid legacy applications running within scientific workflows, our previous work on the OCM-G monitoring system for parallel Grid applications was used [5]. However, the OCM-G alone was not fully sufficient, lacking, among others, mechanisms for fine-grained runtime instrumentation. It was therefore extended in order to support adequate functionality [3,6].

[1] APART, http://www.fz-juelich.de/zam/RD/coop/apart/.

3. Key challenges in Grid workflow monitoring

Scientific workflows running on a Grid have a few important characteristics which make the monitoring of such workflows a challenging task. The Grid scientific workflows can be characterized as follows:

- Dynamic, distributed, and decentralized.
- Loosely coupled in space and time.
- Composed of multiple runtime layers, e.g. services and legacy applications.
- Heterogeneous with respect to their implementation language, infrastructure technologies, and platforms.

Considering those characteristics, we have found the following problems as key challenges in monitoring of Grid scientific workflows (1) handling the inherent and multi-aspect heterogeneity of workflows within a single monitoring infrastructure; (2) supporting on-line collection of workflow monitoring events; (3) monitoring of workflow legacy backends; (4) recording workflow executions, including workflow provenance information. Monitoring of legacy applications within workflows is described in [3] and [6]. The problem of recording of workflow executions has been presented in [4]. This paper focuses on the first two challenges. They are summarized below along with the proposed solutions, which are described in more detail in the following sections.

1. *Addressing workflow heterogeneity*. Multi-aspect heterogeneity of workflows implies that multiple underlying monitoring and instrumentation technologies and concepts may be involved in monitoring of a single workflow. To handle this diversity, the Grid workflow monitoring infrastructure should be designed as a *monitoring framework*. The following design decisions were taken in order to fulfill this goal:
   - Adoption of technology- and platform-neutral interfaces for monitoring and instrumentation.
   - Adoption of Standard Intermediate Representation and a standardized instrumentation service in order to support language- and platform-independent instrumentation of workflows.
   - Design of an open architecture which features standard interfaces to support pluggable data sources and instrumentation tools.
   - Design of a standardized type hierarchy and XML representation of workflow monitoring events.

2. *On-line monitoring support for workflows*. In a Grid environment where the state of resources is ever changing and the applications are often long-running, it is desirable to support on-line collection of monitoring data to enable timely reaction to performance degradation or failures. However, to this end, a fast discovery of workflow activities is required, which is made difficult by the workflow dynamicity, loose-coupling and large scale. In order to address this challenge, *automatic resource discovery* handled by the monitoring infrastructure is proposed, which enables fast discovery of new workflow activities. A distributed data collection algorithm based on a Subscription Coordinator has been conceived for this purpose.

4. Grid workflow monitoring infrastructure

4.1. Architecture

Figure 1 presents the proposed monitoring architecture for Grid workflows and a sample deployment of the monitoring infrastructure. We assume a decentralized architecture where a *Monitor* is deployed at each Grid site, whose responsibilities include exposing monitoring system functionality to end users. This is done via two services – one for monitoring requests (queries and subscriptions), the other for instrumentation requests.

A Monitor manages underlying *Sensors* and *Mutators* which are responsible for actual collection of monitoring events and manipulations of the monitored en-

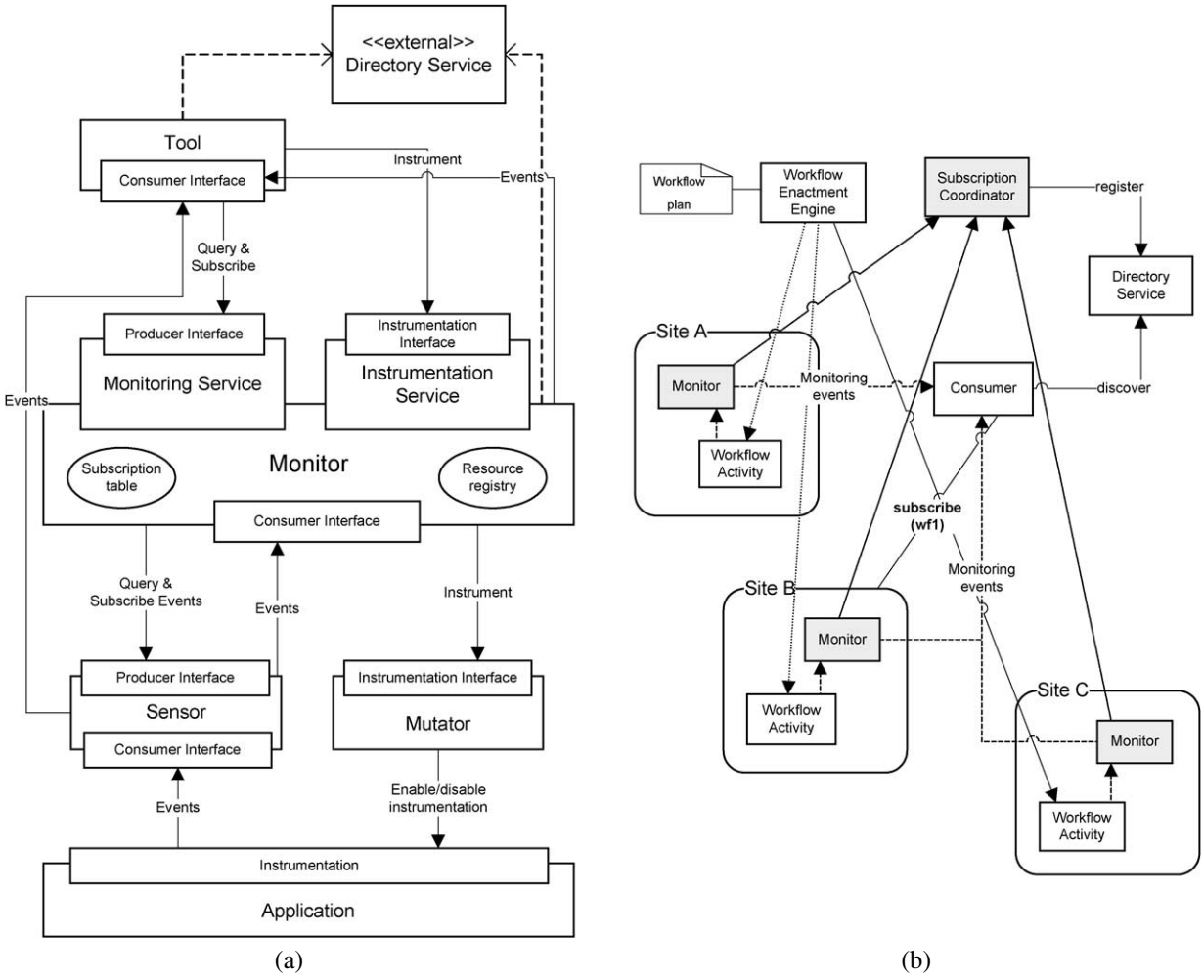

Fig. 1. Monitoring system: (a) architecture, (b) sample deployment.

tities, respectively. Monitoring events are generated by the *instrumentation* placed in the code of applications. The Monitor maintains a resource registry in which underlying monitored resources and their corresponding sensors and mutators are registered. The information about monitored resources (and supported metrics) is published by the Monitors in a *Directory Service* which is used by the consumers to discover appropriate Monitors. The Monitor also maintains a table of active subscriptions in order to forward events from sensors to consumers. Alternatively, the subscription table can be embedded in sensors to enable direct data transfer between producers and consumers.

Sensors and Mutators are collectively referred to as Local Monitors, as they usually reside locally with respect to the monitored resources. Their presence is justified by both functional and non-functional reasons. First, some operations might not be possible remotely, e.g., process manipulations or access to some process-related information may require monitoring agents with access rights equal to the monitored process. Second, the performance concern is related to the monitoring perturbation caused by additional consumption of resources due to monitoring. The division into (global) Monitors and Local Monitors enables the latter to be as lightweight as possible, in order to minimize the monitoring perturbation. In general, there are several performance trade-offs to consider:

- Handling subscriptions can be computationally intensive (e.g. matching monitoring events with requests may involve event filtering, aggregation, transformation or correlation). With separate Monitors (deployed on dedicated machines) to serve this purpose we reduce monitoring overhead but increase monitoring latency due to one additional hop between event sources and sinks.
- Monitoring events generated in the instrumented application could be sent to local or remote monitors (over local or remote communication channels, respectively). In the first case, we reduce the local network traffic but increase CPU consumption due to context switches between the application and the local monitor processes. In the second case, we avoid the context switches, but increase the network traffic and perhaps the latency to send out the events from the application, thus increasing monitoring perturbation.
- Buffering monitoring events in local monitors can reduce local network traffic. Furthermore, buffering the events in the context of the instrumented application can even compensate for the CPU overhead due to context switches, mentioned above. On the other hand, buffering increases memory consumption due to monitoring.
- Processing the monitoring events, such as counting or timing, in local monitors can reduce the network traffic and the utilization of the Monitors, because, instead of full event traces, only aggregated information would be sent. However, it increases CPU consumption due to monitoring. (Though, on the other hand, some other processing, e.g. that related to the execution of network protocol stacks, is reduced.)

On the sample deployment diagram (Fig. 1(b), sensors and mutators are omitted for clarity), a Subscription Coordinator is also depicted. This component supports on-line collection of workflow monitoring data which is described in Section 5.

4.2. Interfaces

There are three interfaces involved in the monitoring system: (1) the producer interface, (2) the consumer interface, and (3) the mutator interface. The producer interface is used to request the monitoring data. This data can be pushed to consumers via the consumer interface. The mutator interface can perform various types of manipulations. A unique feature of our architecture is a dedicated *instrumentation interface* (a kind of mutator interface). A Monitor exposes two services – the Monitoring Service and the Instrumentation Service – which implement the producer and the instrumentation interface, respectively.

The producer interface is based on a producer protocol defined by PDQS (Performance Data Query Subscribe), an XML-based language for specification of monitoring data requests [21]. A monitoring request typically contains the following elements:

- type of request (query or subscribe),
- identification of the resource to monitor,
- monitoring data type,
- subscription period (only for the subscribe request).

PDQS allows to express two types of requests: *query* and *subscribe*. A query for monitoring data synchronously returns the requested data. A subscription request, in contrast, allows to specify a subscription period in which the monitoring data will be asynchronously delivered whenever it occurs, via the consumer interface.

The instrumentation interface realizes the instrumentation protocol defined by the WIRL language

(Workflow Instrumentation Request Language) [21]. WIRL, as PDQS, is an XML-based language which can be used to specify instrumentation requests. An instrumentation request consists of the following elements:

- type of request (get standard intermediate representation, enable/disable instrumentation),
- specification of application's processing unit to which the request should be applied,
- instrumentation specification (for enable/disable instrumentation): which code regions should be affected and the position where the instrumentation should be applied (before or after a code region).

The instrumentation is dynamic and selective, i.e. we can dynamically request to enable or disable the instrumentation for selected parts of the application. In order to enable this, the instrumentation interface allows to obtain an abstract specification of the application's structure expressed as a list of code regions in an XML-based language SIRWF (Standard Intermediate Representation for WorkFlows), an extension of SIR [18]. More information about the instrumentation infrastructure can be found in [6].

4.3. Representation of monitoring data

Low-level application monitoring data is conceptualized in Fig. 2. It is composed of a set of *monitoring events*, each of which is a representation of a *system event*, i.e. an event that actually took place in the application. A monitoring event should contain at least three pieces of information: (1) *event type*, (2) *time stamp*, and (3) *event location*, which describe *what* happened, *when* and *where*, respectively. Event type is an arbitrary description whose meaning is shared between parties involved in monitoring, e.g. a 'function call'. Usually, additional *event data* is needed which is specific for a given event type. It may, for example, specify exactly what happened (e.g. *function name: MPI_Send, parameters: P, result: R*). Location usually describes the application's execution unit (e.g. *Process P, Thread T, Object O, Function F, Code Region C*).

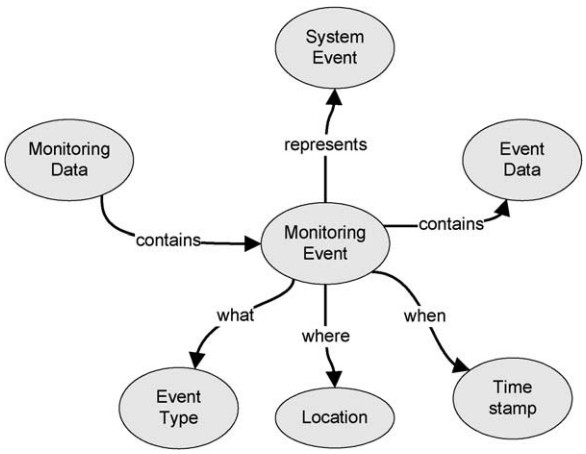

Fig. 2. Conceptualization of application monitoring data.

The following design decisions concerning monitoring data representation have been adopted:

- Event types for workflows have been specified along with the corresponding structures of event data.
- The location information is specified in a portable way by an Experiment Correlation Identifier (see Section 6).
- The events are represented in XML to support loose coupling and interoperability between parties that exchange monitoring data.

4.4. GEMINI: prototype monitoring system implementation

We have developed Generic Monitoring Infrastructure (GEMINI), an implementation of the Grid workflow monitoring architecture. In fact, GEMINI was realized as a framework supporting monitoring of arbitrary entities, in which data sources can be dynamically added as plugins, while the GEMINI infrastructure is responsible for dissemination of resources as well as the collection and transfer of the monitoring data.

Monitoring and instrumentation services in GEMINI are implemented as Globus Toolkit 4 WSRF services.[2] The producer and instrumentation interfaces are also implemented by Sensors and Mutators, respectively. However, they are exposed in a more tightly coupled technology, ICE (Internet Communication Engine)[3] in order to improve performance of data transfers. Also for performance reasons, the consumer interface is implemented in ICE. The monitoring data in the subscription mode is exchanged between Sensors and consumers through a mediating publish/subscribe channel realized as an ICE Storm service which enables event subscription and notification based on topics. The role of the mediator is twofold:

[2] WS-Resource Framework, http://www.globus.org/wsrf/.
[3] http://www.zeroc.com.

1. As with all publish/subscribe channels, to decouple data producers from data consumers.
2. As a data-transfer relay component. Since the monitored data is, due to performance reasons, transfered by means of low-level, tightly-coupled communication channels, communication between producers and consumers across firewalls or private networks may pose a problem. In such cases, the mediator channels could be deployed at a neutral, public location to make the communication possible, at some latency penalty.

The mutators for Java have been developed using the BCEL library which enables to manipulate Java classes.[4] For legacy (parallel) applications, both sensor and mutators have been based on adapting the existing OCM-G monitoring system to the GEMINI framework.

5. On-line collection of monitoring data

In some cases, on-line monitoring is desirable in order to quickly respond to problems. For example, on-line performance monitoring might enable dynamic rescheduling (re-allocation of computational resources) in the case of performance degradation of some computational resources. However, the distributed and dynamic nature of workflows combined with the fully decentralized architecture of the monitoring infrastructure, required in the Grid, makes this task difficult to achieve.

As workflow's activities dynamically emerge at unpredictable time and location, a mechanism of *fast and automatic resource discovery* wherein new producers of workflow monitoring data are automatically discovered and transparently receive subscription requests on behalf of active subscribers, seems to be the key issue to enable on-line monitoring of Grid workflows.

In a Grid environment we cannot assume a fully centralized monitoring architecture (i.e. one wherein all monitoring events would go through a central location) due to scalability problems. On the other hand, the traditional Grid Monitoring Architecture [19] in which all resources are discovered through a global information system, will not be sufficient. The reason for this is that existing Grid information systems are oriented towards query performance, not update performance, and cannot ensure fast notification about new resources. For example, the Berkeley Database Information Index, the information system deployed at over 250 sites in the EGEE project, features a hierarchical architecture in which low-level Grid Resource Information Services (GRIS) provide information to site BDII which is in turn combined and exposed by top-level BDIIs. The information is refreshed in a top-down manner, i.e. a top-level BDII scans site BDIIs which obtain information from GRISes. The refresh is done by reloading the entire database and rebuilding indices *every 2–3 minutes* [1]. Clearly, with such a high refresh rate, this system would not support well online monitoring of workflows.

We therefore argue that the monitoring infrastructure must itself, in part, support automatic resource discovery. Below a solution based on a *subscription coordinator* is proposed (Fig. 1(b)). The coordinator mediates in electing a *main* monitoring data producer for a given workflow. To this end, all producers should first register in the Coordinator. The main producer takes care of subsequent discovery of producers, also using the Coordinator for this purpose. The proposed algorithm is presented in Fig. 3. In this example of monitoring a workflow $wf1$, we have two producers of monitoring data – Monitors $M1$ and $M2$. The scenario also features a *Data Production Coordinator* (Coordinator, $B0$), a Directory Service $D0$, and a Client $C0$. The scenario proceeds as follows:

1. First, Monitor $M1$ registers as producer for $wf1$ in the Coordinator.
2. As $M1$ is the first $wf1$ producer to come, the coordinator elects it as the *main producer* for workflow $wf1$ and registers $M1$ in a directory service $D0$.
3. A client $C0$ which needs to monitoring workflow $wf1$, sends a request to the directory service, and obtains the address of $M1$ as the producer for $wf1$.
4. $C0$ subscribes in $M1$ after which an asynchronous transfer of monitoring events is started, perhaps including some archive events collected before subscription.
5. After a while, a new producer for workflow $wf1 - M2$ – occurs. It attempts to register as a producer in the coordinator but it gets a response that $M1$ is already the one.
6. $M2$ thus registers in $M1$ as a producer for $wf1$.
7. Since $M1$ has a registered subscription from $C0$ it sends a subscription request to $M2$ on $C0$'s be-

[4] http://jakarta.apache.org/bcel.

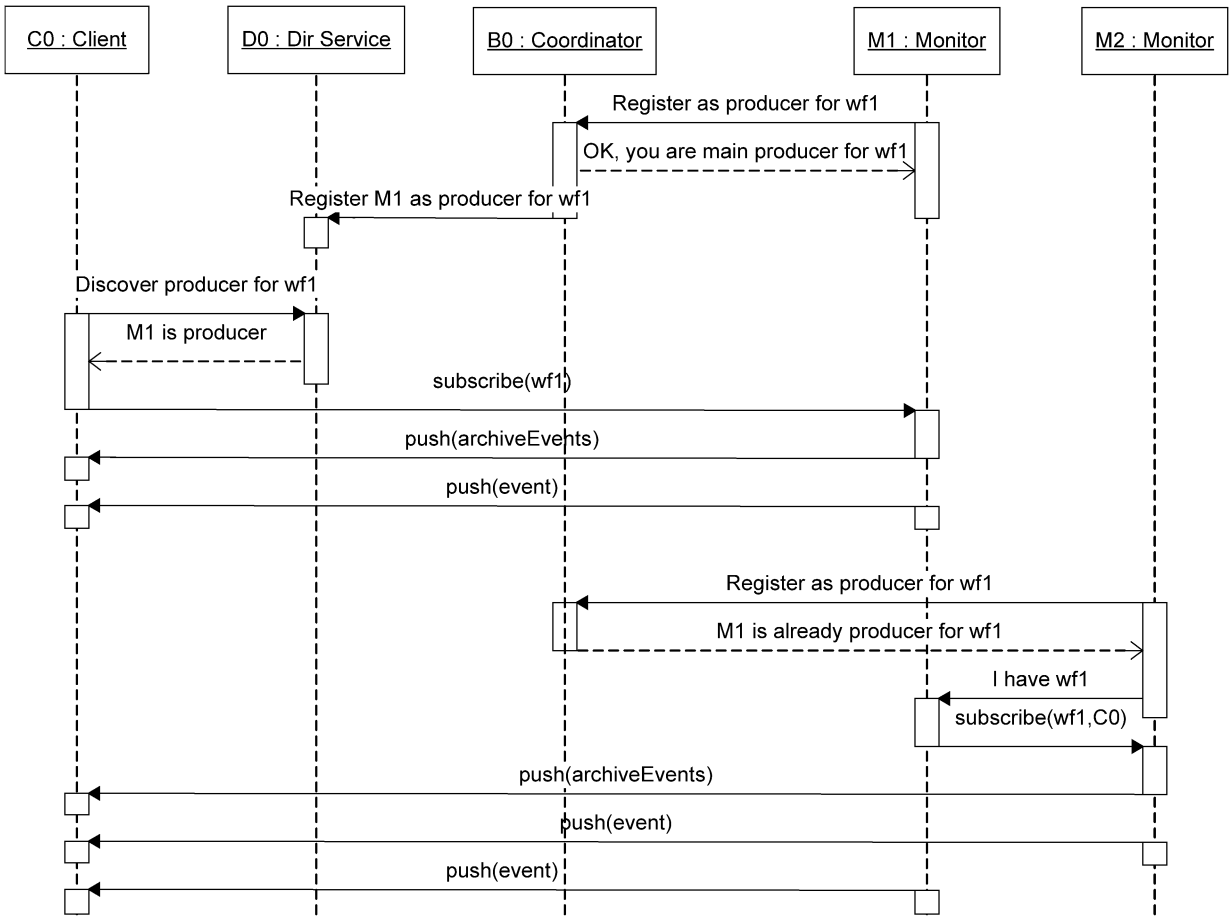

Fig. 3. Data collection algorithm for distributed monitoring.

half. Consequently, $M2$ starts pushing monitoring events to $C0$.

The described algorithm can be characterized as follows. First, the client needs to discover data producers only once, and needs to connect to only one producer. Second, though the Coordinator is centralized, it is very lightweight due to its simple functionality, and it has to handle only arrivals of new producers, not the entire monitoring traffic. Indeed, we have developed a prototype implementation of the Coordinator and performed a model-based performance evaluation which revealed that the proposed monitoring infrastructure can handle workloads as high as around 10 job arrivals per second (this depends on the machine where the Coordinator is deployed). For comparison, the current peak workload in the EGEE project Grid infrastructure reaches 100,000 jobs per day, i.e. not much above 1 job per second.

6. Correlation of monitoring data

The term *correlation* in the context of distributed systems has at least two important meanings:

1. In distributed systems that use asynchronous messaging the problem of matching a request with a response is called the *correlation problem*. The solution is to use a unique *correlation identifier*, which should be the same in the request and in the matching response. This solution is known as the *Correlation Identifier Pattern* ([15], pp. 163–170).
2. In event notification or monitoring systems, the term *event correlation* denotes the process of detecting event patterns (aka *global events*, *composite events*) in a number of seemingly unrelated events. For example, a detected event pattern may reveal a possibility of a security attack.

In the context of workflows, correlation is mentioned in [7]. In general, a process P (i.e. a workflow) may be orchestrated by multiple orchestration engines P_k. The correlation problem is thus, according to the authors, *to associate different parts of process P executing in different P_k*. However, distributed orchestration engines are not of our concern. We focus on a different problem. For monitoring of workflows we shall define the correlation problem as follows:

Workflow monitoring data correlation problem is *to associate different pieces of monitoring data as parts of data related to the same experiment*.

This problem has been, of course, partially recognized. In [13], Grid Workflow Identifiers (GID) are proposed. The GIDs should be created in a "workflow originator" and "propagated to all workflow components". While this is the right way to go, it does not go far enough. The reason for this is that "workflows" described in this work are not *scientific workflows* composed of services and executed by a generic orchestration engine. Rather, a "workflow" is meant as a sequence of different Grid "services" (such as portal, broker, replica manager) whose coordinated cooperation results in execution of a *job*, which in turn does the actual work, and later the results are retrieved.

This model of a 'flat' grid identifier is *not sufficient for monitoring of scientific workflows*, whose execution needs the cooperation of many Grid services as well, but which are themselves composed of many distinct parts, namely activities, and invoked applications, the last often being separate *legacy jobs*, which may well be parallel jobs, again composed of multiple distinct processes.

The correlation of workflow monitoring data must enable association of different pieces of monitoring data at their different levels in the workflow hierarchy, for example events from "*workflow $W0$, activity $A1$*", or "*workflow $W1$, activity $A1$, legacy job $L2$*". Consequently, for the purpose of monitoring we introduce an *Experiment correlation identifier* which is characterized as follows:

1. *Experiment Correlation IDentifier* (ECID) is an extendable XML document which is passed between different parts of executing scientific workflow, as well as Grid middleware services involved.
2. ECID should be included in each event produced by monitoring and it should reflect the *exact space location* of the event in the executing workflow.
3. ECID is *extended on a handover of workflow's execution control* from P_i to P_k, P_i and P_k being different, physically distinct, parts of the executing workflow.
4. Actor responsible for the extension of ECID on handing over the workflow execution to task P_k is the *initiator* I_k of P_k; I_k is normally either the orchestration engine or another workflow task P_i which invokes a subtask P_k.

An example of an ECID for an experiment task's subtask is shown below:

```
<ecid>
  <experiment id="e1">
      <task id="t1">
         <subtask id="s1"></subtask>
      </task>
  </experiment>
</ecid>
```

Experiment correlation identifier is also *important for request-reply correlation*. A client could request a subset of monitoring events that belong only to a certain part of the workflow (e.g. MPI job L_1 invoked from activity A_3). The hierarchical ECID enables the monitoring system to deliver only this subset of events to the client, instead of making the client responsible for filtering.

The technical problem how to pass the Experiment Correlation Identifier to workflow parts remains. Unfortunately, this problem is technology-dependent. In the case of workflow activities implemented as web services this is actually a specific case of a more general problem of *passing context to web services*. Current web service implementations do not support context information in a standard way. For pure, stateless web services, the only information passed to an invoked service is via operation's parameters. There are a few possibilities to pass context information, as discussed in [8], for example:

- by extending service's interface with additional parameter for context information,
- by extending the data model of the data passed to a service,
- by inserting additional information in the SOAP header, and reading it from within the service.

None of those methods is nice and clean. The first one forces to extend the interface of the service which is the least transparent option. The second one is not transparent for the end user, either, and additionally is service-specific. The third one is only possible if ac-

cess to SOAP header is feasible, and it depends on the implementation of the web service container. An additional option is to use the state provided by a service, but this only works for services that support state, e.g. WSRF-based ones.

However, the problem of context-aware web services has been recognized. A Web Service Context Specification has been proposed [23] whose aim is to deal with Web Service context in a standardized way. This specification supports passing the WS context in the SOAP header and defines standardized XML structures to do it.

Therefore, it is natural for us to also follow the SOAP header approach. The workflow enactment engine used in our case – GWES (Grid Workflow Execution Service) [16] inserts additional information to SOAP headers which is accessed in the services (from instrumentation code) to obtain workflow and activity identifiers.

For the legacy jobs, ECID is passed as command-line parameters. However, in the case of MPI applications command line parameters might be passed only to the master process. There are a few possibilities to deal with this. For example, the master process can be instrumented to broadcast the correlation identifier as the first operation after `MPI_Init`. This is the solution we employ.

7. Monitoring example – Coordinated Traffic Management workflow

To demonstrate workflow monitoring, we have chosen the Coordinated Traffic Management (CTM) workflow constructed from application services provided by Softeco Sismat within the K-Wf Grid Project [9]. This application targets the computation of the emission of traffic air pollutants in an urban area and has been developed in tight collaboration with the Urban Mobility Department of the Municipality of Genoa, which provided a monolithic implementation of the model for the pollutant emission calculations, the urban topology network and real urban traffic data. The CTM application workflow has been divided into several different steps in order to allow the semi-automatic composition of services and the definition of a set of ontologies which describe the CTM domain and feed the system with the information needed for the proper selection and execution of services [12].

The main CTM application functionalities are best route, traffic flow and air pollutant emissions calculations. Data graphical representation in SVG format is also supported. For monitoring, a complex use case was used. It consisted of several executable transitions and three control transitions (Fig. 4).

The first activity in the workflow is the generation of a session ID. Next, there are two activities done in parallel – the computation of start and end zone district polygons. Subsequently, node coordinates for start and end are computed, also in parallel. After that, a set of calculations is done for nodes computed in earlier activities. Finally, computations responsible for calculating path length, traffic flow and air pollutants emission follow. Results are also written to the SVG format file, which is done in one of the activities.

During the running of the above scenario monitoring data were acquired by instrumentation of the workflow enactment engine and workflow activities. For each activity, several events were produced: when it was initialized, created, went to the active state, at start and end of the actual running phase, and when it has completed. Each of these events has a precise time of occurrence. Activities such as initialization, creation, activation and completion should be treated as a point in time. The running state is treated as a period of time.

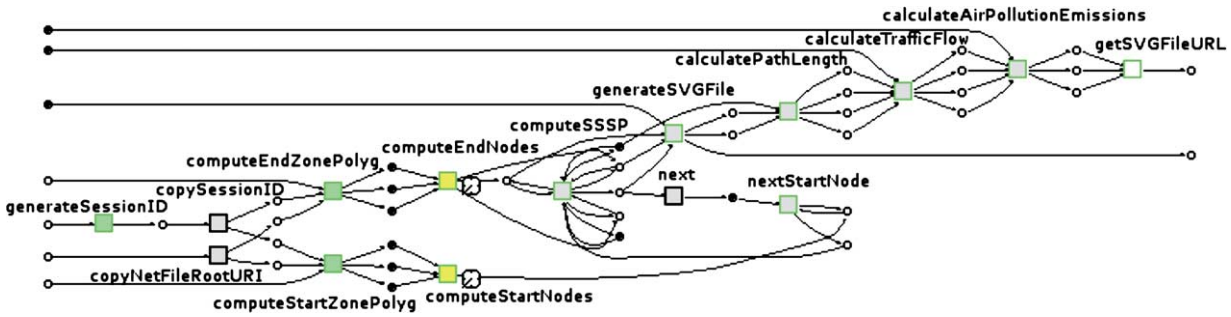

Fig. 4. CTM workflow.

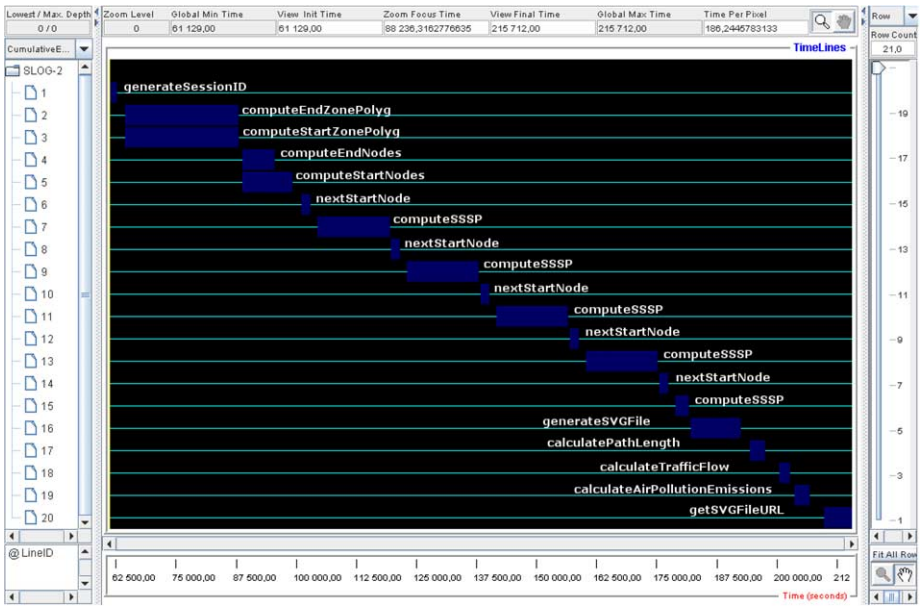

Fig. 5. Monitoring results – global view.

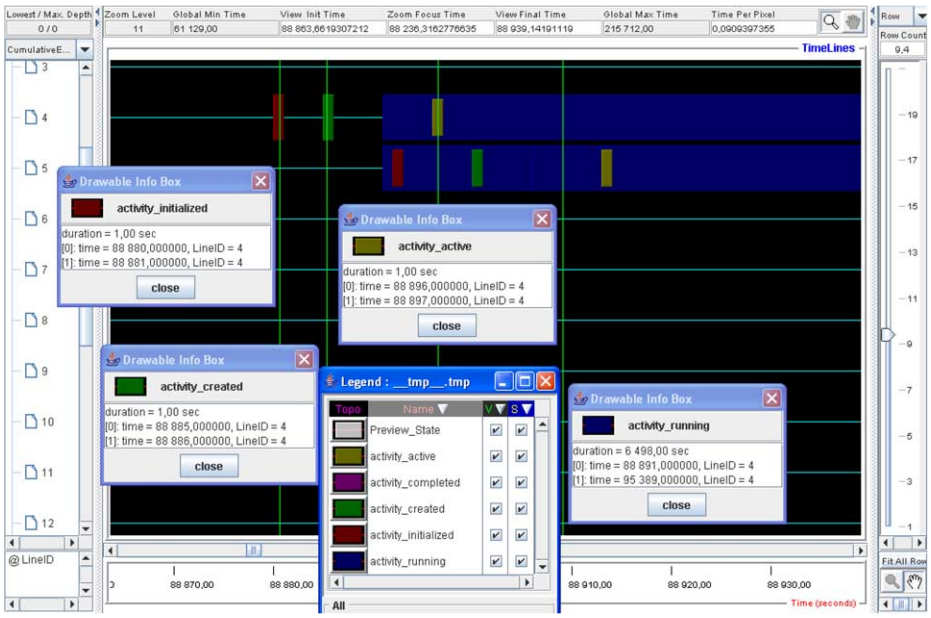

Fig. 6. Monitoring results – detailed local view.

For visualization of the monitoring results, we have used the Jumpshot tool.[5] To this end, events collected from GEMINI were translated to Jumpshot's SLOG-2 format. We can observe how the workflow was executed, how much time each activity has taken, and when it was invoked. Figure 5 presents a global view showing all activities. However, due to the time scale, only the running periods can be seen. Figure 6 presents a zoom of a particular diagram section to show more details – individual events can be seen now. Additionally, windows describing individual bars (representing events and periods) are shown.

[5] See http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm.

8. Summary

We have presented a Grid monitoring architecture for scientific workflows. Specific requirements resulting from the characteristics of Grid scientific workflows were analyzed and an appropriate solutions to deal with workflows' distribution, dynamics and heterogeneity, were proposed. A Grid workflow monitoring architecture has been introduced. It has been pointed out that the traditional Grid Monitoring Architecture needs to be extended in order to support on-line collection of workflow monitoring data. The GEMINI monitoring infrastructure has been presented, and an example of monitoring of a real-life workflow from the K-Wf Grid Project has been shown.

Currently we are working on an ontology-based monitoring data model which can be used to represent a meaningful information about previously running workflows (experiments), and the usage of this information to extract knowledge about the experiments and the computing infrastructure.

Acknowledgements

The work described in this paper is supported by the European Union through the projects IST-2002-511385 K-WfGrid and IST-034364 Gredia, and by the Foundation for Polish Science within the Domestic Grant for Young Scientists.

References

[1] J. Astalos, L. Flis, M. Radecki and W. Ziajka, Performance improvements to BDII – Grid information service in EGEE, in: *Proc. Cracow Grid Workshop'07*, Krakow, Poland, ACC CYFRONET AGH, 2008.

[2] B. Balis, M. Bubak, J. Dziwisz, H.L. Truong and T. Fahringer, Integrated monitoring framework for grid infrastructure and applications, in: *Innovation and the Knowledge Economy. Issues, Applications, Case Studies*, Ljubljana, Slovenia, IOS Press, 2005, pp. 269–276.

[3] B. Balis, M. Bubak and K. Guzy, Fine-grained instrumentation and monitoring of legacy application in a service-oriented environment, in: *Computational Science – ICCS 2006, 6th International Conference*, Reading, UK, May 28–31, 2006, *Proceedings, Part II*, V.N. Alexandrow, G.D. van Albada, P.M.A. Sloot and J. Dongarra, eds, Lecture Notes in Computer Science, Vol. 3992, Springer, Reading, UK, 2006, pp. 542–548.

[4] B. Balis, M. Bubak and M. Pelczar, From monitoring data to experiment information – monitoring of grid scientific Workflows, in: *Third IEEE International Conference on e-Science and Grid Computing, e-Science 2007*, Bangalore, India, 10–13 December 2007, G. Fox, K. Chiu and R. Buyya, eds, IEEE Computer Society, 2007, pp. 187–194.

[5] B. Balis, M. Bubak, M. Radecki, T. Szepieniec and R. Wismüller, Application monitoring in CrossGrid and other Grid Projects, in: *Grid Computing. Proc. Second European Across Grids Conference*, Nicosia, Cyprus, Springer, 2004, pp. 212–219.

[6] B. Balis, H.-L. Truong, M. Bubak, T. Fahringer, K. Guzy and K. Rozkwitalski, An instrumentation infrastructure for grid workflow applications, in: *Proc. OTM 2006 Conferences*, Montpellier, France, November 2006, R. Meersman and Z. Tari et al., eds, Lecture Notes in Computer Science, Vol. 4276, Springer, pp. 1305–1314.

[7] G. Brown and R. Carpenter, Successful application of service-oriented architecture across the enterprise and beyond, *Intel Technology Journal* **8**(4) (2004), 345–359.

[8] S. Brydon and S. Kangath, Web service context information, https://bpcatalog.dev.java.net/nonav/soa/ws-context/index.html, 2005. Accessed 20.06.2007.

[9] M. Bubak, T. Fahringer, L. Hluchy, A. Hoheisel, J. Kitowski, S. Unger, G. Viano, K. Votis and K-WfGrid Consortium, K-Wf Grid – knowledge based workflow system for grid applications, in: *Proc. Cracow Grid Workshop 2004*, Academic Computer Centre CYFRONET AGH, Poland, 2005, p. 39; ISBN 83-915141-4-5.

[10] F. E. Bustamante, P. Widener and K. Schwan, Scalable directory services using proactivity, in: *Supercomputing'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Los Alamitos, CA, USA, IEEE Computer Society Press, 2002, pp. 1–12.

[11] W. Funika, P. Godowski, P. Pegiel and M. Bubak, Towards user-defined performance monitoring of distributed Java applications, in: *E-SCIENCE'06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, Washington, DC, USA, IEEE Computer Society, 2006, p. 54.

[12] T. Gubala, D. Harezlak, M. Bubak and M. Malawski, Semantic composition of scientific workflows based on the Petri nets formalism, in: *Proc. 2nd IEEE International Conference on e-Science and Grid Computing* (available only on CD-ROM), IEEE Computer Society Press, 2006.

[13] D.K. Gunter, K.R. Jackson, D.E. Konerding, J. Lee and B. Tierney, Essential grid workflow monitoring elements, in: *Proc. 2005 International Conference on Grid Computing and Applications, GCA 2005*, Las Vegas, Nevada, USA, Hamid R. Arabnia and Jun Ni, eds, CSREA Press, June 2005, pp. 39–45.

[14] J.J. Hoffman, A. Byrd, K. Mohror and K.L. Karavanic, PPerfGrid: A grid services-based tool for the exchange of heterogeneous parallel performance data, in: *19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, IEEE Computer Society, Denver, CA, USA, 2005.

[15] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, October 2003.

[16] F. Neubauer, A. Hoheisel and J. Geiler, Workflow-based grid applications, *Future Generation Computer Systems* **22** (2006), 6–15.

[17] N. Podhorszki, Z. Balaton and G. Gombas, Monitoring message-passing parallel applications in the grid with GRM and mercury monitor, in: *Grid Computing. Proc. 2nd European Across Grids Conference*, Nicosia, Cyprus, January 2004, Springer.

[18] C. Seragiotto, H.L. Truong, T. Fahringer, B. Mohr, M. Gerndt and T. Li, Standardized intermediate representation for Fortran, Java, C and C++ programs, Technical report, Institute for Software Science, University of Vienna, 2004.

[19] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski and M. Swany, A grid monitoring architecture, Technical Report GWD-PERF-16-2, Global Grid Forum, January 2002.

[20] H.L. Truong, B. Balis, M. Bubak, J. Dziwisz, T. Fahringer and A. Hoheisel, Towards distributed monitoring and performance analysis services in the K-WfGrid project, in: *Proc. PPAM 2005 Conference*, Poznan, Poland, Springer, 2006, pp. 157–163.

[21] H.-L. Truong, P. Brunner, T. Fahringer, F. Nerieri, R. Samborski, B. Balis, M. Bubak and J. Rozkwitalski, K-WfGrid distributed monitoring and performance analysis services for workflows in the Grid, in: *Proc. 2nd International Conference on e-Science and Grid Computing, e-Science 2006*, Amsterdam, Netherlands, IEEE Computer Society, 2006.

[22] H.-L. Truong and T. Fahringer, SCALEA-G: a unified monitoring and performance analysis system for the grid, *Scientific Programming* **12**(4) (2004), 225–237.

[23] Web Services Context Specification (WS-Context) Version 1.0, http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.pdf, 2006. Downloaded 27.03.2007.