Scientific workflows as services in caGrid

DOI:
10.1109/ICWS.2009.19

Document Version
Submitted manuscript

Published in:
2009 IEEE International Conference on Web Services, ICWS 2009|IEEE Int. Conf. Web Serv., ICWS

Scientific Workflows as Services in caGrid: a Taverna and gRAVI approach

Wei Tan[1], Kyle Chard[2], Dinanath Sulakhe[1], Ravi Madduri[1], Ian Foster[1],
Stian Soiland-Reyes[3], Carole Goble[3]

1 Computation Institute, University of Chicago and Argonne National Laboratory, Chicago, IL, USA
2 School of Mathematics, Statistics and Computer Science, Victoria University of Wellington, NZ
3 School of Computer Science, University of Manchester, Manchester, UK
4 Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA

{wtan, foster, madduri, sulakhe}@mcs.anl.gov, kyle@mcs.vuw.ac.nz,
soiland-reyes@cs.manchester.ac.uk, carole.goble@manchester.ac.uk

*This is the **author-submitted version** of https://doi.org/10.1109/ICWS.2009.19*

Abstract

In scientific collaboration platforms such as caGrid, workflow-as-a-service is a favorable concept for various reasons, such as easy reuse of workflows, access to remote resources, security concerns and improved execution performance. We propose a solution for facilitating workflow-as-a-service based on Taverna as the workflow engine and gRAVI as a service wrapping tool. We provide both a generic service to execute all Taverna workflows, and an easy-to-use tool (gRAVI-t) for users to wrap their workflows as strongly-typed services, without developing service code. The signature of the strongly-typed service is identical to the corresponding workflow's input/output definition and is therefore more self-explained to workflow users. These two categories of services are useful in different scenarios, respectively. We use a tumor analysis workflow as an example to demonstrate how the workflow-as-a-service approach benefits the execution performance. Finally a conclusion is drawn and future research plan is pointed out.

1. Introduction

The combination of Web services and scientific workflow technology has made it more convenient for scientists to leverage available data and computation resources, streamlining data processing and exploration, in a Web-scale manner [1]. Existing research activities have offered many scientific workflow systems, to name a few, Kepler [2], Taverna [3], Pegasus [4], SWIFT [5], etc. The popular business workflow language, BPEL, is also adopted by some researchers [6].

caGrid [7] is the service-oriented grid infrastructure for the National Cancer Institute's cancer Biomedical Information Grid (caBIG) program, providing a platform to efficiently share research data to accelerate cancer research. A workflow tool suite is considered an important component of caGrid, as it accelerates service discovery, composition, and orchestration. The features of the workflows in caGrid include:

1. Web scale. Participant services in a workflow are distributed in a wide-area network. caGrid currently consists of 100+ data/analytical services which are provided by institutions all over US, (see caGrid portal: http://cagrid-portal.nci.nih.gov/), and there is no dedicated network to connect them. This means in some circumstances, network bandwidth is limited to interconnect these Grid services.
2. Steps in workflows (mostly caGrid services) are strongly-typed Web services using standardized communication mechanisms. Data exchanged is usually embedded in SOAP messages rather than using proprietary file formats.
3. Moderate size and granularity. Experiences gained from using caGrid services show that, data query/processing service invocation usually takes a moderate time (several seconds to several minutes) and yields moderately sized data sets (kilo bytes to several mega bytes). It is also shown in [8] that queries against biological sequencing data resources often result in *small-to-medium* sized data sets. This is the granularity that Web services are suitable to handle.

These features motivate the idea of wrapping workflows as services for various reasons, such as easier access by users, security concerns, and improved performance. The features presented also differentiate our work with the approaches already adopted by Pegasus, BPEL, etc, in decentralizing the execution of a workflow.

In the following sections we first introduce the motivation of workflow-as-a-service; we then present the architecture and technical approaches to implement this idea with Taverna and gRAVI (Grid Rapid Application Virtualization Interface) [9]; afterwards we present a tumor analysis workflow as an example to analyze how the workflow-as-a-service approach benefits the execution performance; finally we compare our approach with related work and draw a conclusion.

2. Why workflow as a service

In the caGrid project we have adopted and extended Taverna as the workflow solution to orchestrate Grid services which conform to Web Service Resource Framework (WSRF) standard [1]. Feedback from users shows that, besides the capability to orchestrate individual services in a workflow, there are sufficient reasons to expose a caGrid workflow as a service as well:

- Reusability. Users need to reuse workflows as "off-the-shelf" components, without knowing the internal implementation details. Some users are not willing to bother with the installation of a workflow engine before they execute workflows and see the results. In the meantime, users still reserve the option to reveal more details regarding the workflow, like the workflow definition and execution status.
- Accessibility. Some resources may not have built-in support to be remotely accessed; a workflow engine that is local to these resources can be the proxy to access them, and this proxy can access multiple resources in a predefined sequence.
- Security. Users may not have the credentials required to access services individually; however they can be allowed to execute some workflows composed of these services. This means that they are given a group credential allowing the controlled access to these services. This is similar to the idea of using *view* as a as a virtual table in database practice: i.e., hiding the internal database schema while allowing a controlled access to it.
- Performance. With the aid of workflow-as-a-service, a "big" workflow can be divided into multiple parts and co-executed by multiple (distributed) workflow engines (coordinated through their service interface). This "decentralized execution of a workflow" can improve execution performance (as we can see in Section 4 of this paper).
- Interoperability. Wrapping workflows as services also enables the interoperation among multiple workflow systems.

Given the various motivations behind workflow-as-a-service, there are basically two usage scenarios.

Scenario 1: workflows for individual services. When a given workflow is going to be frequently accessed, it is desirable to have a service that serves as the access point for it. The signature of the service corresponds to the input/output definition of the workflow, so that users have a well defined standardized interface. For example, a service can take a "microarray experiment ID" as input and yield the "clustered microarray" complex object, as defined in the workflow.

Scenario 2: a generic service for all workflows. While in scenario 1 each individual service serves the execution capability for one workflow, there are circumstances that an omnipotent service is needed to serve all workflows. This generic execution service has an interface that accepts workflow definitions along with their inputs. With this service deployed in a Grid, users can execute any workflow, or delegate a part of it (i.e., a subflow), to this service, for various reasons like security, performance and data access. This generic service is useful if a specific service for a workflow (or a subflow) is not available. This may be due to the workflow being relatively specific and therefore not accessed much or because no one has developed a service for the workflow. A generic service may also be desirable as it requires only a single deployment and can be used to easily share computing power amongst a group of collaborators. Although this seems to be a panacea to the idea of workflow-as-a-service, it has disadvantages. First it is not as usable by consumers as interfaces are generic to all services rather than being application specific. Moreover, access by consumers is more complex as they must parse the input/output to appropriate file formats and must also have workflow definition files in order to invoke the service

These two scenarios fulfill different requirements and complement each other. Therefore, we propose a framework that facilitates these two scenarios and have developed a tool suite to support both. Our work is based on Taverna workbench which provides a rich set of extension points to enhance its capabilities.

3. Wrapping caGrid workflows as services

To fulfill the requirements of workflow-as-a-service and to realize the two scenarios presented in Section 2, we have designed a solution framework (see Fig. 2) and implemented the required components, the newly developed components are denoted with rounded rectangles. The components in this framework are as follows;

- Taverna workbench: it is not only the workflow modeling tool, but also the execution engine that underpins all workflow processing.
- The workflow-wrapper plug-in: a Taverna plug-in that passes a workflow definition file to gRAVI-t where it is wrapped and deployed. This approach corresponds to scenario 1 presented in Section 2.

- The workflow-execution plug-in: a Taverna plug-in that passes a workflow definition file with its inputs to a weakly-typed generic service where it is executed. This approach corresponds to scenario 2 presented in Section 2.
- gRAVI-t: a extension of our previously-developed gRAVI. gRAVI-t wraps each workflow as a strongly typed WSRF service and Introduce deploys the service into a compliant container.
- Individual workflow services: generated and deployed by gRAVI-t. Each service corresponds to a workflow, and the service's signature is identical to the workflow.
- The generic execution service: a weakly-typed service which can execute all Taverna workflows. This service takes a workflow definition file and its inputs as input parameters, and returns the workflow's execution output.
- Globus ws-core: Globus ws-core 4.0.3 is used as the service container because 1) it is the container used for other services in caGrid 2) the WSRF specification supported by Globus can be used to expose various workflow properties such as definition, workflow status, etc.

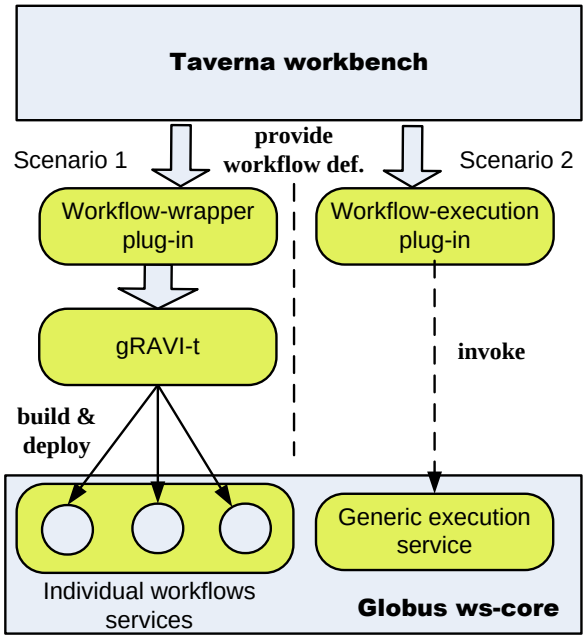

Fig. 1 Solution framework for workflow-as-a-service. Individual services, which can be built and deployed with gRAVI-t on-the-fly, are strongly-typed and each corresponds to a specific workflow. Generic execution service is a weakly-typed one that can execute all Taverna workflows.

We discuss the realization of these two scenarios in Section 3.1 and 3.2, respectively. It is important to note that both generic and individual typed services can be consumed by any Web service client (which is not shown in Fig. 1) including Taverna workbench.

3.1 Approach 1: workflows for individual services

Having created a workflow in Taverna workbench a user may wish to expose the workflow as a service, to do this the user simply starts the workflow-wrapper plug-in which passes the workflow definition file to gRAVI-t. gRAVI-t parses the workflow definition and generates a Web service with a method *execute* to start the workflow. The signature of this method is constructed by using the input/output ports defined in the Scufl definition file used by Taverna. That is, each input port corresponds to an input parameter and each output ports corresponds to an output method.

gRAVI (grid Remote Application Visualization Interface) is a plug-in extension to Introduce [10] which is aimed at wrapping binary applications as secure WSRF compliant Web services. The major focus of gRAVI has been in providing a wrapping toolkit that does not require service developers to write any implementation code, description files, or deployment scripts. The resulting WSRF service is encapsulated in a standard GAR file which is able to be deployed to any compliant container without the requirement for any gRAVI or Grid infrastructure. All creation and deployment steps can be performed in the Introduce GUI, removing the need for users to run scripts or create/modify definition files. gRAVI-t builds upon gRAVI by adding the ability to create a service from a workflow definition rather than wrapping a binary executable.

gRAVI-t generated services make use of a primary service to invoke the workflow using the Taverna API and a secondary *context* service to create and manage instance resources used to retrieve output and monitor execution. When the primary service is invoked the input parameters are collated in the appropriate format and the workflow is started, a resource is created to represent the invocation and is used for subsequent monitoring and control of the workflow execution. The context service has methods to retrieve typed workflow output, monitor the process through explicit polling or WS notifications, retrieve standard output, errors, and files created by the workflow and destroy the resource.

A unique individually typed service is created for each definition file which can then be invoked any number of times whilst storing information relating to the individual invocation. The definition file is packaged with the service GAR file so the service contains all information required to deploy and run remotely. When the service is invoked the workflow is executed by the local Taverna and a reference to the resource is returned to the client. When the invocation finishes the output files are grouped

using a unique UUID for each invocation, clients can retrieve the results by accessing the appropriate typed output retrieval method (there is one method per output defined in the workflow), in the future we plan to dynamically form complex XML objects that will encapsulate all output ports in a single object and could then be retrieved in a single method.

The process of creating a service using gRAVI-t is relatively straight forward. The standard Introduce service creation wizard guides the user through setting up the service skeleton including the service name, namespace, location, and java package. The user then selects the gRAVI-t extension and is presented with the gRAVI-t dialog to select the specific workflow definition. From here the service is completed creating the strongly typed interface and includes the workflow execution components and monitoring methods via the resource. Having completed the creation process the Introduce GUI can also be used to deploy the service to an appropriate container. In this process the user does not need to write any scripts, definitions files, or code in order to create and deploy a fully functional strongly typed WSRF Web service.

3.2 Approach 2: a generic service for all workflows

Instead of wrapping a single workflow to a service, we also provide a generic service to which any Taverna workflow can be submitted for execution. This service is generic because it can virtually execute any Taverna workflow. The interface of the service is as follows:

```
String [] executeworkflow (
  String workflowDefinition,
  String [] inputPortNames,
  String [] inputValues,
  String [] outputPortNames)
```

In this interface definition, *workflowDefinition* is the Scufl definition file (in a string format) of the Taverna workflow to be executed. *inputPortNames* is an list of input port names. *inputValues* is a list of input values that correspond to the sequence given in *inputPortNames*. *outputPortNames* is an list of output port names, and the return value of this interface is a list of output values that correspond to the sequence given in *outputPortNames*.

This generic service can be used in a number of situations. To list a few:

- To execute a workflow using this generic execution service, when users want to leverage the power of a Grid but a specific service for this workflow is not available.
- To outsource a subflow to the Grid. In this case the main workflow is still executed in Taverna workbench; however, a subflow inside the main flow is delegated to a Grid service that provides execution capability. Which subflow is delegated, and which execution service to delegate to can be chosen independent of the workflow definition.

These two approaches meet different demands and complement each other; our framework shown in Fig. 1 facilitates both. In the next Section we will outline the performance benefit of using workflow-as-a-service.

4. Applications

In this section a tumor analysis workflow from caGrid is used to demonstrate how the workflow execution can benefit from the idea of workflow-as-a-service. Among the various benefits we mentioned in Section 2, this section presents a quantitative analysis on the performance improvement brought about by workflow-as-a-service.

4.1 Tumor treatment analysis workflow

A tumor treatment analysis workflow [11] from caGrid describes the experimental procedure to extract and analyze gene expression of tumor tissues to find groups of genes associated with efficacy of treatment with a new medicine. Patients enrolled in this clinical trial are treated with a new medicine. Results show that some tumors respond, and some do not.

This workflow contains three major stages to determine the underlying biomolecular interactions that yield this result. (See Fig. 2.)

- First, obtaining bio-specimen data from both responding and non-responding patient groups. In this step, a patient subset is identified and two groups of biospecimens are selected based upon the treatment response. That is, we query the caTissue service to obtain two groups of tissues: the responder and the non-responder. Moreover, in this step we need to confirm that the biospecimens obtained contain malignant tissue so that they are appropriate to be used for analysis in next steps.
- Secondly, analyzing gene expression data to identify genes that are expressed differentially in responders vs. non-responders. Using the information contained in tissue data, further queries are sent to the caArray service to retrieve the microarray data, and then a statistical method called *t-test* [12] is used to identify genes that have significantly different expression levels between the responders and non-responders.
- Finally, identifying potential biomarkers. Using the differentially expressed genes identified in t-test analysis, the pathways are retrieved using the Pathway Interaction Database (PID). This pathway contains a list of genes and the known interactions; more genes may be added and some eliminated

depending on the identified pathways. These pathways help to understand the biomolecular interactions and key cellular processes in cancer treatment.

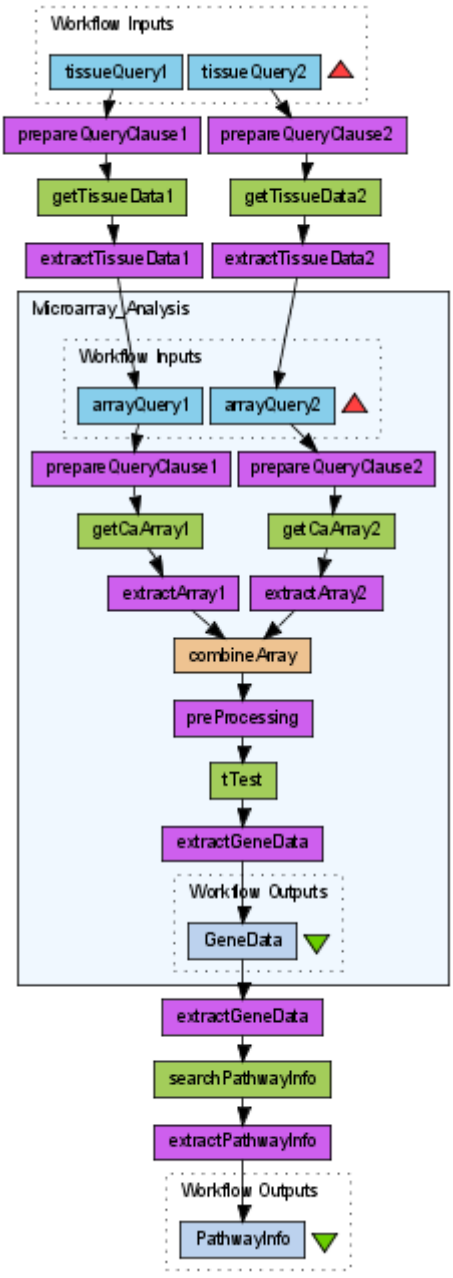

Fig. 2 Tumor treatment analysis workflow. The Microarray_Analysis subflow is highlighted and is to be executed as a service since it is data intensive.

By examining the topology of this workflow, the location of participant services and the volume of data exchanged between them, we identified that the microarray analysis subflow is suitable to be outsourced to speed up the execution. This subflow is highlighted in Fig. 2 with the name *Microarray_Analysis*. The volume of its input (*arrayQuery1*, *arrayQuery2*) and output (*GeneData*) data is small (the input query clauses are usually ~10KB size, and the output is usually 1/20 of the size of the fetched gene array data) compared to the data exchange within the workflow (from ~100KB to 100MB) – that is because using the array query clause, a relatively large data set is retrieved and sent to t-test, t-test compares the data and returns a much smaller subset of genes that are differently expressed. Moreover, usually the caArray service and the t-test service are collocated and the Taverna workbench, as the client, accesses them remotely via a WAN. Therefore, a natural assumption is that, if a workflow execution service which is local (which means accessible via LAN) to the caArray and t-test services, is used to execute the identified subflow, the execution time of the whole workflow will be reduced. This hypothesis is validated through experiments outlined in sections 4.2.

4.2 Experiment design

The design of the experiment is as follows. To preclude the performance fluctuation of caGrid services in its production environment, instead of using services in a production Grid, we built our own services that will be composed into the tumor workflow. These services have no real data querying or processing capabilities but yield an output of a given size upon request. This design keeps the workflow's topology and data flow volume "real", allowing experiments to isolate and examine the workflow execution efficiency, i.e., not influenced by the load of the services and their processing time.

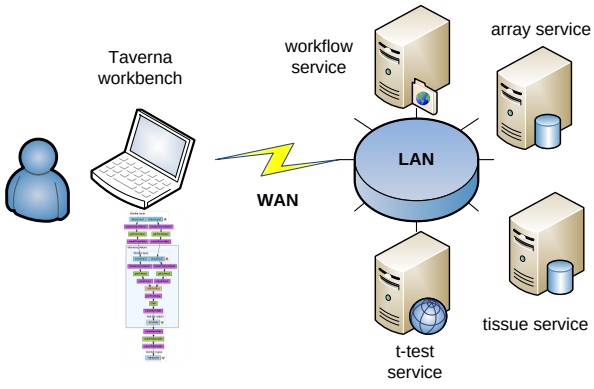

Fig. 3 Environment of the performance experiment. Scientists start workflows in Taverna workbench that connects to caGrid through WAN. In the mean while the services in the microarray subflow are connected via a high-speed LAN.

The network environment of this experiment is shown in Fig. 3. The participant services, i.e., the array, the tissue and the t-test service are deployed in Linux servers. A strongly-typed workflow service is deployed to a Windows server to execute the microarray subflow exclusively. These servers are connected through a gigabyte Ethernet LAN. In all servers Globus ws-core 4.0.3 is used as the service container. A client side Taverna workbench is used to initiate tumor analysis workflows, and this workbench connects to those services through WAN. The server configuration for this experiment is shown in Table. 1.

Table.1 Server configuration for the performance evaluation experiment.

Purpose	CPU	RAM	OS
Taverna workbench	Intel Dual Core 2.26G	4G	Windows
Microarray workflow service	Intel Dual Core 2.26G	2G	Windows
Participant services	2*Intel Dual Core 2.80G	3.0G	Redhat Linux

Tumor analysis workflows are configured in three ways, i.e., execution mode, data load, and WAN bandwidth. The configurations of data load and WAN bandwidth simulate various possible scenarios under which caGrid workflows are orchestrated. As a consequence, under these configurations, the execution times in different modes are good measurements to demonstrate the advantages of workflow-as-a-service.

Execution mode

Tumor workflows are executed in two different modes, i.e., central mode and subflow-as-a-service mode, based on whether to use the workflow service or not.

Central Mode (**CM**): the whole workflow is orchestrated by the embedded engine inside the workbench. This engine is remote (i.e., connect via WAN) to all participant services. This is the most common use scenario for scientific workflow systems.

Subflow-as-a-service Mode (**SM**): the main workflow is orchestrated by the embedded engine inside the workbench, while the subflow is orchestrated by the workflow service which is local (i.e., connect via high-speed LAN) to the participant services.

Data load

The microarray data queried from each biospecimen varies from 20KB to 25MB, and therefore the data exchanged inside this subflow ranges from 80KB to 100MB, and the t-test results are configured to 1/10 of the size of the microarray data from one biospecimen.

WAN bandwidth

As we have pointed out in Section 1, caGrid is a web-scale grid infrastructure and there is no dedicated network connection among participant services. To simulate this network property, we adopted three types of bandwidth for the WAN connection shown in Fig. 4, i.e., 1Mbps, 10Mbps, and 100Mbps.

In our performance tests, we focus on the execution time of individual runs. For each combination of execution mode, data load and WAN bandwidth we run the microarray workflow 20 times and use the average as their execution time. A collective summary of the execution times of the tumor workflow in CM and SM, under different data load and WAN bandwidth settings, is shown in Fig. 4. Three figures represent the execution times under three WAN bandwidth settings (1Mbps, 10Mbps, and 100Mbps), respectively. The horizontal axis of each figure represents the data loads under which workflows run, and the vertical axis represents the execution time.

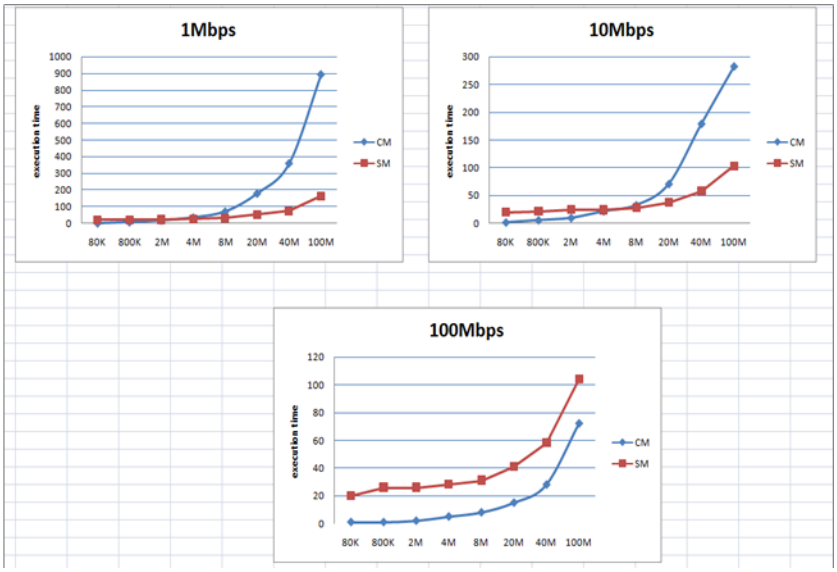

Fig. 4 Execution time comparison of the Tumor workflow, in different modes (central mode, CM and subflow-as-a-service mode, SM), data load (from 80KB to 100MB) and WAN speed (from 1Mbps to 100Mbps).

4.3 Result analysis

Fig. 4 verifies our hypothesis that, SM performs much better than CM, when the ratio of data load to WAN bandwidth is big. Under 1Mbps and 10Mbps bandwidth, the execution times in SM increase much more gradually than those in CM. This result confirms that, given the nature of the tumor workflow, the idea of workflow-as-a-service can reduce the network traffic in WAN and ultimately speed up the overall execution.

The break-point at which SM substantially exceeds CM (when we say "substantially exceeds" we mean that using SM doubles the execution speed) rises from 8MB at a 1Mbps bandwidth, to 40MB at a 10MBps bandwidth. At a 100MBps bandwidth, without a surprise, CM exceeds SM in all data loads we examined. However as we mentioned earlier, since caGrid spans all over the US and there is no dedicated network connection between services, and services do not likely to offer their whole bandwidth to a single user, it is not likely that such a high speed bandwidth is available for a workflow execution.

Meanwhile, when the ratio of data load to WAN bandwidth is small, SM is not worthwhile due to the overhead brought about by outsourcing the orchestration of the microarray subflow to a service. Besides the one-time effort of developing and deploying a workflow service (either a strongly-typed specific service or a generic service) there is overhead when compared to executing a workflow in a single local Taverna workbench. The overhead of executing a workflow (or a part of it) as a service includes: 1) the transfer of data and workflow definition file (if using the generic service) to the execution service; 2) overhead brought about by the Web service protocol stack, such as SOAP message encoding and parsing; 3) starting the Taverna API inside the workflow service, to parse the workflow file, the participant services and setup for execution.

The services used in our experiment are non-functional services, that is, they only feed data but there is no processing component (like querying or analyzing data). Therefore, the execution times examined in these experiments do not include data processing time which would occur in actual service invocation. Our experience with caGrid shows that the actual data processing time in each service invocation varies from seconds to minutes, so the improvement brought by workflow-as-a-service is significant enough for the performance of these fine granularity workflows. (Given the data load and network bandwidth in our experiments, the performance improvement might not be significant for workflows with long jobs that take hours/days to finish.)

The experiments also show that, if biology data centers can collocate a workflow service with their data and analytical resources, orchestrating highly repetitive routines which are data intensive, this could alleviate the data transfer over a wide-area network and in turn speed up the execution of scientific workflows. Moreover, workflow execution service in Grid has more processing power and can alleviate the burden of the Taverna workbenches running on scientists' desktops.

5. Related work

We divide related research efforts into three categories, i.e., workflows as services, decentralized BPEL execution, and optimization of scientific workflow execution.

Workflows as services

A service based approach to integrate multiple engines to co-execute workflows is proposed in [13]. A submitter service called GEMLCA is developed to submit workflows to four types of engines (P-GRADE, Taverna, Triana, and Kepler). Our approach is more comprehensive because we not only provide a generic service for all Taverna workflows, but also provide gRAVI-t to wrap individual workflows as strongly-typed services, which is more practical for end users. Moreover, our service is WSRF compliant and can provide more information regarding the workflow. In the current implementation our service can provide workflow status as resource properties, and adding more workflow metadata as resource properties of these services is in our plan.

Decentralized BPEL execution

In the business process management (BPM) domain, decentralizing a workflow into multiple partitions and assigning each to one engine for co-execution is also desirable due to security and performance concerns. For example, researchers from IBM India Research Lab have conducted a series of studies on decentralized execution of BPEL processes. This work includes model partition method, fault-handling in a decentralized manner, and performance improvement [14], dataflow constrained partition [15], and optimum decentralized orchestration with limited network bandwidth [16].

As for the performance evaluation, we focus on the execution time of individual runs, instead of the throughput index in [14]. The reason is, in BPM it is common that a workflow will have many concurrent instances representing different transactions (for example, in an online-shopping company, an order processing workflow may have thousands of concurrent instances), and this is not going to happen in a Taverna workbench installation on a scientist's desktop. Therefore our focus is

on reducing the execution time of each single run. Compared to the BPEL processes shown in [14], the data load of the caGrid workflows is much larger (up to 100MB rather than several KBs).

Especially, we need to differentiate our work with BPEL based approaches. BPEL processes are inherently Web services and workflow-as-a-service comes for free. However, our approach is unique in terms of:

- As we have discussed in our previously published paper [17], compared to BPEL, Taverna is a more dataflow-oriented system that better fulfills the requirements of modeling the workflows in caGrid. Moreover, because data flows are explicitly modeled and there are no implicit flows (like BPEL), Taverna workflows are more modular and easy for decentralized execution. A subflow can easily be replaced as an execution service.
- Our service is WSRF compliant and can expose various workflow metadata as resource properties. We have added workflow status as a resource property, and can add other resource properties when required by users. gRAVI-t also provides an option of using file staging if input/output of a service is really big and files are used rather than embedded XML content.

Optimization of scientific workflow execution

Eliminating-the-middleman [18] is an approach that inside a workflow, data exchange among services is not through the engine but through proxies local to the services. Experiments show that this architecture results in substantial performance improvements. Our workflow service can be seen as a type of data proxy which stores result data of service invocation. But our service is more than a data proxy since it also orchestrates the services in a predefined sequence.

Pegasus [4] is a popular scientific workflow system in e-science. It uses a layered framework that hides the complexity of infrastructure from end users. A workflow compiler will receive an abstract workflow and plan it into a concrete one based on available data and computation resources. This concrete workflow is later executed by DAGMan workflow engine. The planning phase embeds a lot of optimization effort, in terms of data access, task clustering, workflow partitioning and job scheduling. Pegasus workflows are usually made up of non-service applications (i.e., jobs) which often use files as inputs/outputs. A Pegasus workflow usually runs in a cluster or grid environment which is not as widespread as caGrid. The nature of Pegasus workflow and its hosting environment is quite different from Taverna workflows by nature. However, the ideas behind the approaches adopted by Pegasus are good stimulus to our work. For example, the idea of workflow-as-a-service is similar to workflow partitioning and task clustering in Pegasus, although our approach and infrastructure are quite different.

6. Conclusion

In scientific collaboration platforms such as caGrid, workflow-as-a-service is favorable for a number of reasons, such as reuse of workflows, accessing remote resources, security concerns and the improvement of execution performance. We provide a solution for workflow-as-a-service based on Taverna, gRAVI, and their extensions. We not only provide a generic service for all Taverna workflows, but also an easy-to-use tool (gRAVI-t) to wrap individual workflows as services. These two categories of services are useful in different scenarios, respectively. We have shown the performance improvements gained using the workflow-as-a-service approach by presenting a real world tumor analysis workflow. This approach significantly speeds up the workflow execution compared to the single engine approach when the ratio of data load to WAN bandwidth is big, by reducing the network traffic in WAN.

Although we believe the idea of workflow-as-a-service is promising, there are many enhancements that could be made. We plan to tackle these issues for future work:

- It is now possible to outsource a portion of a workflow to another engine via the workflow service, but which part(s) to be outsourced is a complex problem and factors such as location of service/data/workflow service, bandwidth and security boundary are to be considered.
- When workflows are executed as services, intermediate data is stored in the service's site. A catalog and fetch mechanism is needed if end-users need to use them.

7. Acknowledgements

This project has been funded with Federal funds from the National Cancer Institute, National Institutes of Health, under Contract No. N01-CO-12400.

References

[1] W. Tan, I. Foster, and R. Madduri, "Combining the Power of Taverna and caGrid: Scientific Workflows that Enable Web-Scale Collaboration," *Internet Computing, IEEE,* vol. 12, pp. 61-68, 2008.

[2] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice & Experience*, 2005.

[3] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice & Experience*, vol. 18, pp. 1067-1100, 2006.

[4] E. Deelman, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, pp. 219-237, 2005.

[5] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: Fast, Reliable, Loosely Coupled Parallel Computation," in *2007 IEEE Congress on Services*, 2007, pp. 199-206.

[6] B. Wassermann, W. Emmerich, B. Butchart, N. Cameron, L. Chen, and J. Patel, "Sedna: A BPEL-Based Environment for Visual Scientific Workflow Modeling," in *Workflows for E-science: Scientific Workflows for Grids*, I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, Eds.: Springer-Verlag, 2007, pp. 428-449.

[7] J. Saltz, S. Oster, S. Hastings, S. Langella, T. Kurc, W. Sanchez, M. Kher, A. Manisundaram, K. Shanbhag, and P. Covitz, "caGrid: design and implementation of the core architecture of the cancer biomedical informatics grid," *Bioinformatics*, vol. 22, pp. 1910-1916, 2006.

[8] H. Stockinger, T. Attwood, S. N. Chohan, R. Cote, P. Cudre-Mauroux, L. Falquet, P. Fernandes, R. D. Finn, T. Hupponen, and E. Korpelainen, "Experience using web services for biological sequence analysis," *Briefings in Bioinformatics*, 2008.

[9] K. Chard, C. Onyuksel, W. Tan, D. Sulakhe, R. Madduri, and I. Foster, "Build Grid Enabled Scientific Workflows using gRAVI and Taverna," in *SWBES08: Challenging Issues in Workflow Applications Workshop*, Indianapolis, USA, 2008.

[10] S. Hastings, S. Oster, S. Langella, D. Ervin, T. Kurc, and J. Saltz, "Introduce: An Open Source Toolkit for Rapid Development of Strongly Typed Grid Services," *Journal of Grid Computing*, vol. 5, pp. 407-427, 2007.

[11] D. Messersmith, "Life Sciences Use Case," 2008.

[12] P. Baldi and A. D. Long, "A Bayesian framework for the analysis of microarray expression data: regularized t-test and statistical inferences of gene changes," *Bioinformatics*, vol. 17, pp. 509-519, Jun 2001.

[13] T. Kukla, T. Kiss, G. Terstyanszky, and P. Kacsuk, "A general and scalable solution for heterogeneous workflow invocation and nesting," in *Third Workshop on Workflows in Support of Large-Scale Science*, 2008, pp. 1-8.

[14] G. B. Chafle, S. Chandra, V. Mann, and M. G. Nanda, "Decentralized orchestration of composite web services," in *13th international World Wide Web conference*, New York, NY, USA 2004, pp. 134-143.

[15] G. Chafle, S. Chandra, V. Mann, and M. G. Nanda, "Orchestrating composite Web services under data flow constraints," in *2005 IEEE International Conference onWeb Services* 2005, pp. 211-218 vol.1.

[16] G. Chafle, S. Chandra, N. Karnik, V. Mann, and M. G. Nanda, "Improving Performance of Composite Web Services Over a Wide Area Network," in *IEEE Congress on Services*, 2007, pp. 292-299.

[17] W. Tan, P. Missier, R. Madduri, and I. Foster, "Building Scientific Workflow with Taverna and BPEL: a Comparative Study in caGrid. ," in *4th International Workshop on Engineering Service-Oriented Applications (WESOA'08)*, Sydney, Australia, 2008.

[18] B. Adam, B. W. Jon, and H. Jano van, "Eliminating the middleman: peer-to-peer dataflow," in *17th international symposium on High performance distributed computing* Boston, MA, USA: ACM, 2008, pp. 55-64.