Strengths and Weaknesses of Sub-Workflow Interoperability

Beth Plale, Eran Chinthaka Withana, Chathura Herath, Kavitha Chandrasekar, Yuan Luo, Felix Terkhorn
School of Informatics and Computing
Indiana University, Bloomington, Indiana, USA
{plale, echintha, cherath, kavchand, yuanluo, masterkh}@cs.indiana.edu

***Abstract*—Workflow orchestration tools have gained recognition for their role in scientific discovery. As the number of workflows proliferate, reuse and reproducibility become issues of importance. Reuse of a workflow is enhanced when the system on which it runs guarantees execution. An increasingly prevalent of reuse however, begs for interoperability between workflow systems. We carried out a comparative evaluation of the practicality of workflow interoperability by focusing on sub-workflow interoperability and evaluating different systems within that form. The approach shows surprisingly uniform performance, relegating the tradeoffs to qualitative differences.**

I. Introduction

Workflow systems are an integral component of cyberinfrastructure for e-Science [9] [3]. The e-Science workflow is often modeled as a task graph, made up of tasks often loosely coupled and coarse-grained. Workflow orchestration draws from human and business processes, for instance scheduling manufacturing operations, inventory management, and business process management [8], and these applications of workflows focus on optimizing for efficiency because they are run repeatedly over a long timeframe between changes. In the e-Science domain, however, efficiency is traded off for flexibility and dynamism of a workflow system in the face of an ever-advancing science discovery process.

Workflow systems often provide default activities that can be used as components of a workflow. These activities may be domain independent, such as third party data movement, or targeted towards a particular domain such as a BLAST gene sequence matching activity. Workflow systems can be categorized by their interaction with and assumptions about back end compute resources. A system might be targeted to work with a back end Linux or Windows cluster, run workflows on the user's workstation, submit jobs to a Grid, TeraGrid [6], or a cloud platform. As the size of the back end resource grows, the workflow system supporting it provides additional constructs for large-scale parallel execution of jobs.

An increasingly prevalent of reuse however, begs for interoperability between workflow systems. A 2007 NSF/Mellon sponsored workshop [23] explored the topic; it was subsequently advanced by Elmroth et al. [12] who suggest that interoperability can take several forms, expressed as three Levels. In Level 1 interoperability, workflow system is able *coordinate workflow tasks designed for another workflow system*. Level II interoperability or *sub-workflow interoperability* is where sub-workflows are shared between systems. The third level of interoperability, Level III, is *complete workflow interoperability* where it is possible to execute a workflow designed for one system by another.

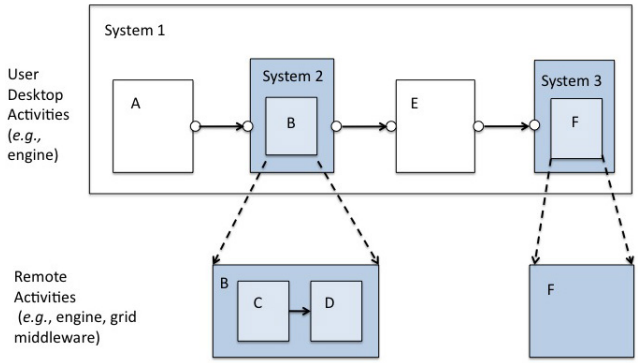


Fig. 1. Sub-workflow interoperability shares workflows between systems in this diagram adapted from [12]. Activities B and F are called from System 1 by instructing Systems 2 and 3 respectively to execute the activities. B, while seen as a single activity in System 1, is actually a subworkflow. F is an activity that runs on grid middleware.

We posit that of the three forms of workflow interoperability, sub-workflow interoperability is likely to have the longest lasting value. The reason is as follows. Most reuse of workflows in myExperiment.org [16] is of sub-workflows, as has been noted by its authors, that is, users share portions of workflows, and these sub-workflows are being picked up for adoption at higher rates than are full workflows. Suppose further then that a workflow system guarantees reproducibility of its sub-workflows (or full workflows). If a new user were to include the sub-graph into her workflow, it is reasonable then that she would be inclined to run the subworkflow where it is guaranteed to run. Suppose now that the researcher has two such sub-workflows whose guarantees are provided by two different workflow systems. It is reasonable then for her to create a single workflow that uses the two component workflow systems where the guarantee is strongest.

While we posit that sub-workflows will have longest lasting value, to our knowledge there is no good comparative data on the costs, both quantitative and qualitative of adopting the strategy. We undertook to fill in the gap through a performance evaluation of sub-workflow interoperability that fixes the high level system at a user desktop workflow engine, and explores the performance and programming impact of various forms of

remote activity. Adhering to the categorization and terminology of Elmroth et al. 2010 and shown in Figure 1, System 1 is hosted on a user's desktop machine. In this model, workflow activities run where the engine runs. This is the case for activities A and E. Two forms of sub-workflow interoperability are shown in the figure. Activity B is called from System 1 by instructing System 2 to execute the activity. B, while seen as a single activity in System 1, is actually a subworkflow consisting of activities C and D. Activity F is called from System 1 by instructing System 3 to execute the activity. F, which is seen as an activity in System 1 calls out to grid middleware to carry out the execution of the activity. Other forms of sub-workflow interactivity can exist, but a system that can utilize local machine resources for simple execution and use remote resources for more complex tasks is simpler in the simple case. Remote access workflow systems are complex distributed systems, and that programming complexity should not hurt the simple case, enforcing the adage what a user doesn't know should not hurt them. In our evaluation we hold System 1 constant as a user desktop workflow system, specifically, the Trident Scientific Workflow Workbench [5]. Trident was chosen because it is easy to use and as a Windows desktop solution could benefit from sub-workflow interoperability as there is a preponderance of scientific functionality running on Linux based systems. Trident uses the .NET Workflow Foundation for workflow execution.

We undertook a qualitative and quantitative evaluation of sub-workflow interoperability. Using the model in Figure 1, for the System 2 case we evaluate the performance of the Kepler workflow system [4], and the Apache ODE [1] workflow tool. For the System 3 case where remote services are invoked directly we evaluate GFac [22] and the Opal toolkit [24]. We also evaluate each system's Model of Computation (MOC) [12] with respect to the host workflow system. The model of computation, according to Elmroth, considers the definition of interaction between activities. Intuitively, the MOC gives interpretation to the edges that connect two vertices of a workflow graph (i.e., the activities). We are interested in determining how compatible a remote system MOC is to the local system MOC with respect to typing, control or data flow, and scheduling of activities.

The contribution this paper makes is a careful study of the tradeoffs of a sub-workflow interoperability solution. The remote engine model compares favorably to the remote grid middleware in terms of performance.

The remainder of this study is organized as follows. Section II discusses related work. Section III details architectural aspects. Section IV discusses the workload and Section V discusses the quantitative and section VI qualitative results. Section VII discusses future work.

II. Related Work

A wide range of workflow systems [29] [18] have been developed for scientific research. While systems such as Dagman [28] are general purpose [29], others have a stronger domain adoption [26]. Some of these systems are designed to optimize for supercomputing resources (e.g. Grids [29], Dagman [28], Wings [15]) while others are better suited to desktop or single user environments such as Taverna [26]. Even among general purpose workflow systems, some have unique capabilities, making them the ideal choice for certain applications. For example, in domains where parametric sweeps are common Pegasus [10], in combination with DAGMan [14] treats parametric sweeps as first class operations and provides an ability to handle very large number of jobs.

Because of the large number of workflow systems and their unique capabilities scientists are researching ways to standardize different workflow systems. Jinde et al. [20] is a study of interoperability amongst business workflows management systems. There have been efforts to standardize workflow interoperability and Hayes et al. 2002 [19] presents the standards the workflow systems need to adhere to achieve business workflow interaction, yet these standards imposed limitations on workflow systems to cater for requirements that arise at a particular business environment. The AFRICA [30] framework proposes interoperability among workflow systems which use SOA based XML message interactions. Even though some of these efforts are promising, different workflow representations (e.g. directed graphs, petri-nets, UML diagrams), unique domain and compute resource specific requirements within different workflow environments have complicated attempts at interoperability. Certain workflows systems [26] [21] provide capabilities for Level I interoperability using Web services based activities/actors.

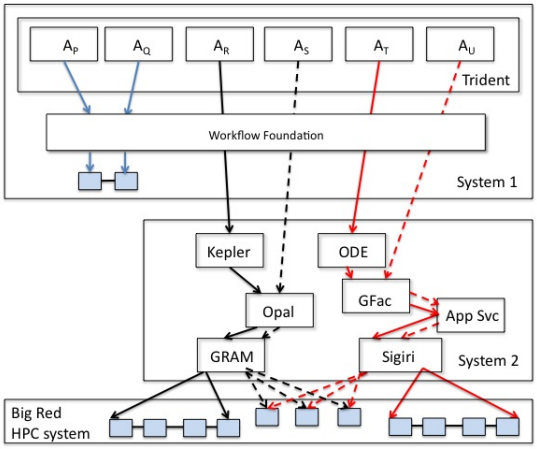


Fig. 2. Types of sub-workflow interoperability: local execution within system 1 is shown by activity A_P and A_Q invoking a workflow through WorkflowFoundation. Activity A_R communicates with the Kepler remote engine to run a subworkflow on Big Red (solid black lines). Activity A_S contacts grid services directly to invoke nodes individually (dotted black lines). Activity A_T invokes the ODE workflow engine to run a subworkflow on Big Red (solid red lines). Activity A_U contacts grid services directly to invoke nodes individually (dotted red lines).

III. Architectural Organization

The architectural organization presented here defines the infrastructure used in the study. It can be seen as a combination

of cases that might be used in practice. While exercising the whole architecture is unrealistic in a real setting, the sub-workflow cases individually and in small-group combinations are realistic. Too, the architecture clearly demonstrates the components held constant in the study, that is, we held the top level workflow framework to be a desktop workflow system. There are four primary components of the architecture, which are refinements to the model of Figure 1 and these are:

Baseline execution: The baseline execution does not use sub-workflow interoperability, and instead runs workflows locally, within System 1 in Figure 2. The top level workflow engine is the Trident Scientific Workflow Workbench and it runs workflows locally. Trident calls out to the Workflow Foundation to orchestrate execution.

Remote workflow engine: In the remote workflow engine case, an activity in System 1 contacts a remote workflow engine, illustrated by activities A_R and A_T in the figure. A_R invokes Kepler which contacts the Opal Toolkit to execute a subworkflow on the Big Red supercomputer through Globus Gram. Activity A_T invokes the Apache ODE workflow engine that contacts GFac to execute a subworkflow on Big Red using Sigiri resource manager [7].

Remote grid/cloud middleware: A second form of sub-workflow interoperability is illustrated by activities A_S and A_U. These activities contact grid/cloud middleware directly. Activity A_S contacts Opal. Since there is no orchestration at the remote system, the remote grid/cloud services are capable of executing only one task of a workflow. Shown in the black dotted lines is GRAM executing one of the three services of a sub-workflow Similarly activity A_U contacts GFac for execution of one of the three tasks pointed to by the dotted red lines.

High performance compute resources these are the high performance compute resources such as Teragrid, OpenScienceGrid, or a supercomputer.

Using terminology of Figure 1, the top level workflow system is the desktop engine Trident Scientific Workflow Workbench. Trident runs on a Windows machine. The local workflow case, or baseline case, does not use sub-workflow interoperability. Its entire workflow, depicted in Figure 2 as Trident activities A_p and A_q invoking two workflow tasks under the control of Workflow Foundation running on the same host or host cluster as Trident.

The second case is illustrated by activity A_R's invocation of the Kepler workflow system to orchestrate a workflow. Kepler uses the resource manager, Opal, to submit jobs to the supercomputer, Big Red; Opal submits using Globus Gram [13]. This case illustrates sub-workflow interoperability where a local workflow engine must interact with a remote workflow engine (as does the case illustrated by A_T). The third case is A_S initiating a remote grid/cloud service directly. It exercises the case of sub-workflow interoperability achieved by means of direct invocation of grid/cloud resources as categorized by Elmroth et al. 2010. Cases 4 and 5, A_T and A_U, are second examples of cases A_R and A_S respectively. A_T invokes the Apache ODE workflow engine which schedules jobs using GFAC, and WS-GRAM [13] with job communication facilitated through a wrapper service that allows a web services infrastructure to communicate with legacy Fortran codes. A_U invokes remote grid/loud services directly.

IV. Workflow workloads

The core workflows of the study cover four cases over two workflow patterns as follows:

1) Data Intensive, Sequential Workflow
2) Data Intensive, Parallel Workflow
3) Compute Intensive, Sequential Workflow
4) Compute Intensive, Parallel Workflow

The sequential workflow is made up of 5 identical tasks executed sequentially. For the compute-heavy version Linpack Java version is used. Linpack solves linear equations, and we configured it to carry out execution on a 5000 x 5000 matrix, using double precision floating point coefficients. The inputs and outputs are small, on the order of 5KB. The data-intensive workflow carries out applies a cryptographic hash function (i.e., MD5) on a 1.5 GB file. Because there are no natural data dependencies in this workflow, a 1.5 GB file is copied from one location to another before each MD5 service is performed. The parallel workflow executes the same computations (MD5 or Linpack) in parallel. The workflow is sandwiched on either end by scatter and gather nodes.

A sequence diagram shown in Figure 3 temporally depicts activity along a vertical timeline of invocation sequences for the Kepler/Opal stack, showing communication between workflow engines and grid services.

V. Evaluation

The experimental performance evaluation piece of our study is intended to reveal the performance overheads of adopting a sub-workfow solution. The metrics used to determine key overheads are

1) Time difference, Td_i, to invoke workflow instance i on local machine versus time to invoke workflow through a remote workflow engine. That is, the time penalty of using a second workflow engine.
2) Difference between sub-workflow through a second workflow engine versus directly accessing remote grid services.
3) Variability in approaches for subworkflow stacks. This gives the variation for a given service time or latency. The best and worst case can often vary significantly, leaving the user uncertain of which will perform better on any given day.

Test environment. The Trident Workflow Workbench and Workflow Foundation run on a Windows server, Intel E7450 four 2.0 GHz 24-core processor, 128 GB RAM running Windows Server 2008 R2 Enterprise edition. We used Trident version 1.2.1 and Workflow Foundation 4. The remote services, shown as System 2 in Figure 1, are run on a cluster consisting of 16 dual-socket, 2 core (4 total cores/node) MD Opteron system with 16 GB of memory per node running

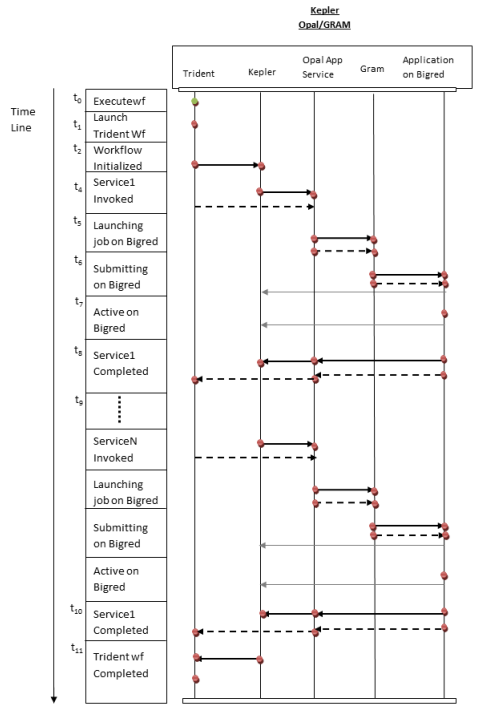


Fig. 3. Linear sequence diagram of sub-workflow interoperability. Remote engine interaction between Trident and Kepler. Solid lines are sub-workflow events. Dotted lines are remote grid services events. Gray lines are notifications.

64-bit Red Hat Enterprise Linux. Nodes are connected via Gigabit Ethernet. Local disks are 73 GB in size and nodes in the cluster have GB Ethernet connectivity to a 24 TB NAS that is front-ended by a file server consisting of NFS v4.

The remote workflow tasks are executed on Big Red, a 1024 IBM JS21 Blade server, each with two dual-core PowerPC 970 MP processors, 8GB of memory per node. Compute nodes are connected via gigabit Ethernet to a GPFS file system hosted on 16 IBM p505 Power5 systems. We submit jobs to run on the system through IBM's LoadLeveler resource manager. LoadLeveler, in turn, relies on Adaptive Computing's Moab scheduler to dispatch user jobs to appropriate and available compute nodes. In our experiment, we use an experimental queue of BigRed which has 4 node/16 core job capacity but very low queue latencies.

The data intensive workflows make use of the underlying Luster Distributed File System for file movement within the same site. The workflow suites used in the evaluation are capable of moving files between remote sites using GridFTP and RFT, but the data intensive workflows in this evaluation focus on data movement within the same site. Every activity in the Data Intensive Sequential Workflow copies the data file to its input working directory before it starts processing and once it finishes the processing copies the output file to its output working directory. So every activity in Data Intensive Sequential Workflow will copy its input file from the output working directory of the previous activity. The first activity would copy the input file from the input parameter to the workflow. The Data Intensive Parallel Workflow is launched with the location of the data files as the inputs to the workflow. Each activity in the workflow copies the input data files to its input working directory and produces its output files to its output working directory. The last activity of the Data Intensive Parallel Workflow gathers all the outputs from the parallel activities to a single output directory. Each test is executed five times and minimum, maximum and median values are computed.

The secondary workflow systems that we use are Apache ODE workflow engine and the Kepler workflow system. For the former, XBaya [27] mediates between Trident and the Apache ODE workflow engine. ODE uses GFac to manage interactions with application services. GFac can instantiate workflow tasks; it moves data for Grid resources and interacts with job submission and resource management providers to schedule jobs to Grid resources. We set up our experiment so that GFac uses Sigiri to submit jobs to the supercomputing resource at IU. For the latter, Kepler workflow engine uses Opal actors to interact with job submission components. We use GRAM actors to submit and manage jobs with the supercomputer to execute jobs. Kepler invokes an Opal actor which consists of a generic web service client to launch job and query job status. The Opal actor submits a job to the Opal server which submits the job to the resource manager in BigRed using GRAM. Once a job is done, the output data is staged from BigRed to Opal Server. When the Opal job status is COMPLETE, the actor will be terminated.

The remote grid services we use are Gfac and Opal. Trident interacts directly either with GFac or Opal server to get the jobs scheduled and executed. Trident also uses a custom file movement utility, implemented as an Opal service, for the file movement to the supercomputer resources, in case of Opal-based workflows. In case of GFac/sigiri remote grid services, GFac serves as data mover.

A. *Baseline of sub-workflow overhead*

We capture an overall measurement of sub-workflow overhead by running a workflow directly within the Trident workstation and comparing that against the same workflow executed by a secondary workflow engine. In order to have a fairer comparison, we ignore waiting time in the job queue in the remote case.

For the compute-intensive workflows, the remote ODE workflow engine version had only 2.4% higher execution time than local execution. The remote grid service version using Gfac/Sigiri, on the other hand, had 15.5% higher execution time than local machine execution. This difference is because of the overhead in invoking GFac, the service factory, for dynamic web service creation. Kepler sub-workflow and Opal/GRAM grid services showed only 5% higher execution time than local machine execution. Hence, the difference in

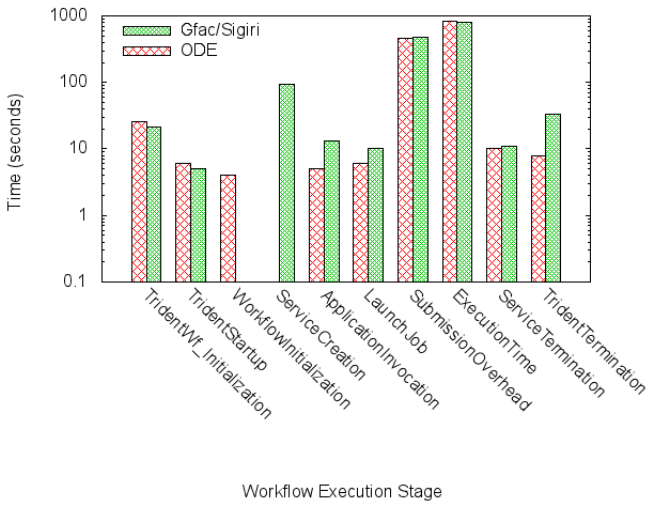
time can be attributed to the difference in architecture between local and subworkflow and remote grid execution. The 10.5% higher overhead for Gfac/Sigiri versus Opal/Gram can loosely be seen as the overhead of a web services approach.

B. Workflow engine and remote grid service options

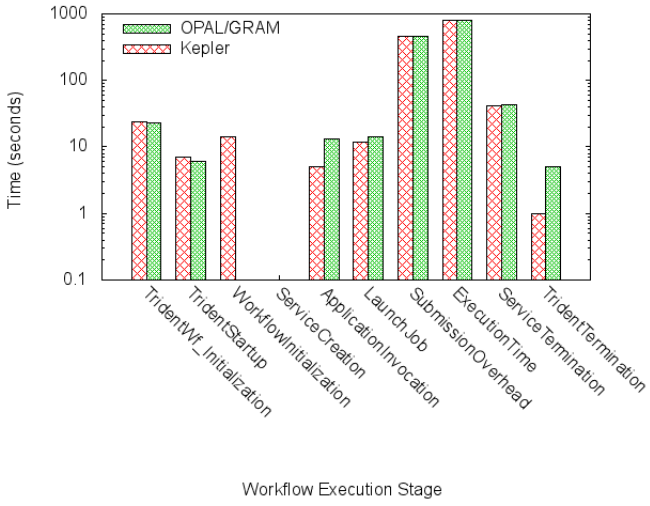
The two remote approaches differ in that one, which we call ”remote engine” has Trident working with a secondary workflow engine. The second approach has Trident executing a ”remote task” directly. Figures 4(a) and Figure 5(a) compare performance of ODE workflow engine option to the remote grid service option (through GFac and Sigiri) for sequential compute intensive and data intensive workflows, respectively, with log scale along Y axis. The performance of the workflow engine stack is slightly better than the remote task case. The remote task stack is dominated by Service Creation and Trident Termination latencies. The ODE stack has the flexibility to reuse a dynamic service. But in the remote task case, Trident must get GFac to dynamically start a new service for every task invocation. This contributes to ”Service Creation” being a significant overhead. Too in the remote task approach, Trident has more activities to manage whereas in the remote engine approach, the number of nodes in the Trident workflow is minimal. This higher number of workflow activities within Trident contributes to the latency captured as Trident Termination. Since ODE uses an internal data movement tool (*i.e.*, GFac and every component in the synthetic workflow consumes the same input file, the tools move the file for the first activity and reuse the same location for the other nodes in the workflow. But for the remote task approach, files are moved for every service invocation adding more overhead to the workflow execution.

Figure 4(b) and Figure 5(b) compare the performance of Kepler workflow engine to Opal/Gram remote task approach for both sequential compute and date intensive workflows. Unlike GFac which create services dynamically, Opal uses predeployed web services and hence Service Creation overhead exists neither in Kepler-Opal subworkflow nor Opal related remote grid service model. In the remote task approach, Trident has more activities to manage and track, whereas for the remote engine approach, the number of nodes in the Trident workflow is minimal.

For the data intensive workflow, data movement is explicitly managed by a Kepler actor in the remote engine case or a Trident activity in the remote task case. Both Kepler's actor(node) and Trident's activity(node) are Opal web service clients. The number of actors/activities in the data intensive workflow doubled due to the data movement services. We put the Execution time of data movement service that happened on execution sites as the Data Received time; and the small submission overhead of the data movement service (due to forking) is added to the Submitting Overhead. Moreover, in Opal stage-out each web service outputs data from the remote execution sites to the Opal server. Hence Kepler/Opal Service Termination time is larger than the ODE/GFac case. Similarly, the higher number of workflow activities within Trident contributes to the latency captured as Trident Termination.



(a) ODE and GFac/Sigiri: compute intensive workflow



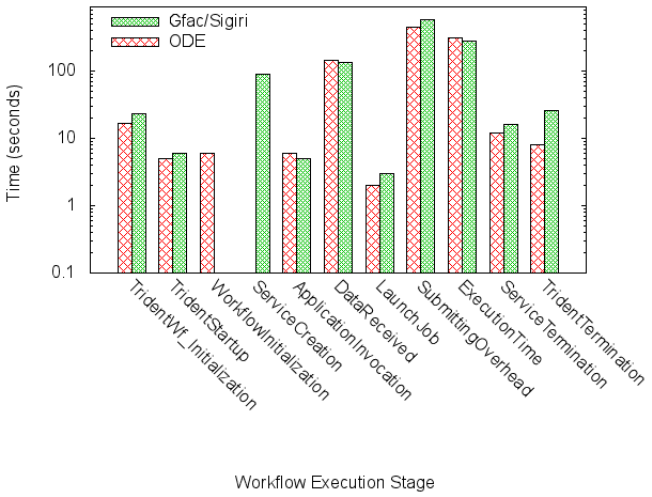
(b) Kepler and Opal/GRAM: compute intensive workflow

Fig. 4. Compute intensive workflows

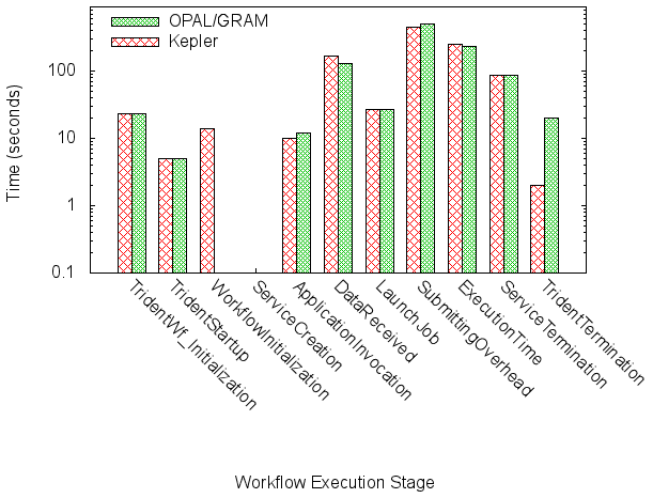
C. Execution variability

Performance of a system is determined in part by the variability of its performance over repeated runs. Figure 6 captures variation in execution times for ODE and Kepler solutions under sequential compute and sequential data workflows. The error bars represent the logarithm of minimum and maximum values of a given measurement and the plotted value represent the logarithmic median of the measured values.

In the case of the sequential compute intensive workflows ODE and Kepler stacks take approximately the same amount of time (16 sec). But examining the deviation between minimum and maximum values, the ODE stack has a smaller deviation (106 sec) to Kepler's (1481 sec). The large deviation of the latter can be attributed to submission overheads. The ODE stack keeps the submission overhead within a small range. Kepler uses GRAM as its job submission middleware and GRAM takes a varying amount of time to get the job



(a) ODE and GFac/Sigiri: data intensive workflow



(b) Kepler and Opal/GRAM: data intensive workflow

Fig. 5. Data intensive workflows

scheduled and executed in the compute resource. During this experiment we also experienced job failures caused by different GRAM failures and had to re-run our experiments on multiple occasions. Additionally, the "Application Invocation" overhead in ODE is significantly lower than other stacks. Higher "Application Invocation" overhead in Kepler could be attributed to inefficient workflow activity and service handling within the Opal actors. For data intensive sequential workflows the ODE and Kepler stacks show similar performance numbers (80 sec). Kepler shows the lowest deviation (575 sec) with ODE at 650 sec. Kepler shows significant overhead during the shutting down phase of the service (Service Termination). Further investigation on the issue revealed that this is due to the implicit log file movements occurring in Opal server.

VI. Qualitative Aspects of Hybrid Model

We experimentally evaluated two models of remote execution for a handful of scenarios to illuminate the latencies and variabilities inherent in different approaches. In this section we discuss qualitative aspects of the comparison specifically:

1) The model of computation (MOC) - the MOC of a workflow system captures such aspects of a workflow system as the execution models supported by a system, the node scheduling algorithm, and typing restrictions on edges.
2) Level of control at desktop
3) Architectural coherence: A measure based on the number of components, fragility of interfaces, etc. which are related to the downstream maintenance and sustainability costs.

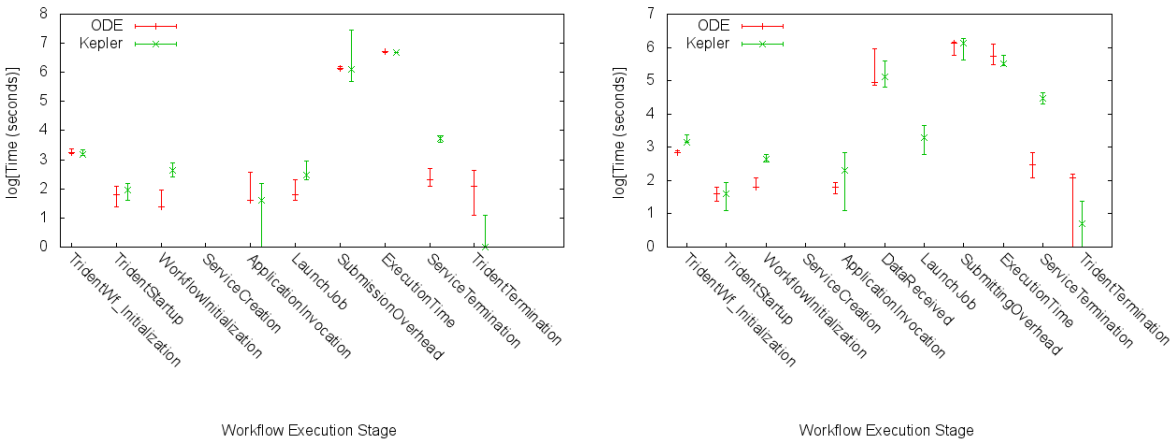
A. Model of Computation

The MOC defines the expressiveness of a workflow system. That is, what kinds of execution models are supported? Does a system support parallel execution? Control flow or data flow execution of DAGs, finite state machines? The model of computation is discussed in Elmroth et al., and the MOC of several systems differentiated in Goderis et al. [17]. In this study we fix the top level workflow system as the Trident desktop workflow engine, and study various forms of sub-workflow interoperability. Further, Elmroth et al. states that "sub-workflows are seen as black boxes and their internal MoC is not important to workflows higher up in the hierarchy", meaning that we need not consider the internal edges of the subgraph (sub-workflow). The black-box nature of the sub-workflow model has advantages and disadvantages. The advantage factors into the picture when one examines the MOC of Trident. Trident is a control-flow workflow system. All scheduling decisions are based on static information and this information is used to generate a Actor/activity firing schedule in the form of a Windows Workflow Foundation runtime script before it starts execution.

Goderis et al. define a process network model where each actor executes in a Java thread, and all actors execute concurrently. Trident does not natively support the process network model. There is only one thread executing when a Trident workflow is executed. As a result, in case of a parallel workflow in Trident, there is no concurrent execution, but only interleaving execution of activities using the same thread. However, Trident can support a process network model by each parallel activity to spawn a child thread, then define a reduce or join type of process that waits for completion of these concurrent asynchronous threads. This is how we executed the parallel workflows of this study. The fact that the sub-workflow is a black box means that the Trident MOC neither extends nor limits the MOC used within the sub-workflow.

B. Level of Control at Desktop

Workflow engines are often bound to certain services to carry out tasks used during workflow execution. These tasks



(a) ODE and Kepler: compute intensive workflow, sequential

(b) ODE and Kepler: data intensive workflow, sequential

Fig. 6. The variability in execution times for the two remote engine approaches can be seen by the error bars of each execution step.

include data movement services, authentication and authorization frameworks, job submission and monitoring services. This significantly limits the functionality of the workflow engine, because it can only perform the kinds of data movement it supports. But through the remote grid model, a user can use any data movement framework she chooses and also has the ability to optimize data movements and data placement in compute resource.

If the top level workflow engine supports checkpointing and recovery operations, when a workflow fails at a certain node, the workflow can be restarted from that node. But if the failed node represents a multiple task sub-workflow, the failure can take time to recover. For example, if a certain experiment contains 10 tasks to be executed and if it is implemented as set of components, then a failure at the 8th node is not excessively costly, because the workflow can be restarted to run at the 8th node. But if the same experiment is implemented as a sub-workflow, and if 3rd to 8th component are in the subworkflow, failure of the 8th node requires the workflow to be restarted from the 3rd task.

The ODE workflow engine, particularly through its instantiation in the OGCE tool suite [2] is limited by its ability to add new workflow activities. For functionality to be added, it must be exposed as a Web service. Both Kepler and Trident workflow engines on the other hand are primarily targeted to desktop based workflow executions and thus provide flexibility to incorporate new functionality easily into the system. With the actor model in Kepler and activity model in Trident, a user can program any functionality into the workflow, enabling a wide variety of functionality to be supported in the workflow.

In experience gained in the research lab and in the classroom, Trident is easy to use. Programming new workflow activities requires writing a small C# activities. With its user-friendly interface and programming model, we believe the tool could appeal to the scientist who is comfortable with a Windows platform and inclined to write and deploy new workflow functionalities that execute within their local compute resources. From the workflow engines with which we have experience, Trident has the lowest time to get a custom workflow up and running.

C. Architectural Simplicity

The remote task approach (over remote engine approach) gives the user more control over the execution of the experiment. But with this control comes maintenance overhead and complexity, because the user has to manage all the components and their interactions. For example, if any of the interfaces that the components are interacting changes, the workflow author has to change the components to adapt to the changes. Whereas in the remote workflow engine solution the changes will be handled inside the calling workflow engine infrastructure. From our experience with the LEAD [11] science gateway, grid middleware adds complexity to the architecture. Handling security issues is also a known concern with these systems. This work becomes more complex with the reliability issues of these middleware components [25]. When Trident is expected to interact directly with Sigiri and Opal, the workflow author will have to handle the complexities for each activity, including authentication mechanisms, fault tolerance and checkpointing. In the sub-workflow workflow engine approach, the user must provide perfect configuration parameters for the next workflow engine to function properly. As we discussed in section V, the cost of failure of a sub-workflow can be higher compared to the component workflow. This gap further widens proportional to number of nodes in the sub-workflow.

VII. Conclusion

The results of our study reveal that the remote workflow engine approach generally outperforms the remote task approach. This is due in part to the fact that the workflow engines we used are currently in use in scientific experiments and have many optimizations built into them. For example, the dynamic service creation and life cycle management within

ODE stack through the generic factory service, significantly reduces the overhead in handling application service invocations. Intelligent file movement services have additionally substantially reduced overheads in data intensive workflows. Additional conclusions include the following:

Grid middleware is responsible for a significant amount of overhead in a scientific workflow stack scheduling jobs into super-computing resources. The job failures and the higher variation of overheads we experienced during our evaluation suggests that the instability and unpredictable behavior of grid middleware components have high impact on the scientific workflow systems that are using them. However efficient and optimized these workflow stacks are, the issues in middleware can make these workflow suites uncertain, if not unusable.

Similar to other workflow systems, Trident facilitates workflow runs in Windows based environments. But we think more improvements are necessary to make this toolkit more useable among the scientific research community. For example, Trident lacks the support for parallel execution constructs within it. Even though activities are picked up and scheduled in parallel, for a parallel workflow, they are executed sequentially.

Scientific applications written using a wide variety of programming and scripting languages, must be wrapped and exposed using interoperable methods for them to be invoked by workflows engines. In our evaluation we used the generic service factory of GFac and Opal to wrap applications. GFac with its current capabilities can expose an application deployed in a supercomputing resource. But with the increased usage of cloud computing platforms for scientific workflow executions we intend to expand our evaluation to include cloud computing platforms using other application wrapping methods.

VIII. Acknowledgements

This project was funded in part by the National Science Foundation under grants NSF CSR-0720580 and NSF EIA-0202048, and from a gift from Microsoft.

References

[1] Apache ode (orchestration director engine). Http://ode.apache.org/.

[2] The open grid computing environments portal and gateway toolkit. Http://www.collab-ogce.org.

[3] *A report from the US national science foundation blue ribbon panel on cyberinfrastructure*. IEEE Computer Society, May 2002. ISBN 0-7695-1582-7.

[4] I. Altintas, C. Berkley, et al. Kepler: An extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pp. 423–424. IEEE, 2004. ISBN 0769521460. ISSN 1099-3371.

[5] R. Barga, J. Jackson, et al. The trident scientific workflow workbench. *IEEE International Conference on eScience*, 0:317–318, 2008.

[6] C. Catlett. The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility. In *ACM International Symposium on Cluster Computing and the Grid*. Published by the IEEE Computer Society, 2002.

[7] E. Chinthaka, S. Marru, et al. Sigiri: Towards a light-weight job management system for large scale systems. Tech. Rep. TR681, School of Informatics and Computing, Indiana University, Bloomington, Indiana, 2009. Http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR681.

[8] E. Deelman, D. Gannon, et al. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009. ISSN 0167-739X.

[9] E. Deelman and Y. Gil. Managing large-scale scientific workflows in distributed environments: Experiences and challenges. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, p. 144. IEEE, 2006. ISBN 0769527345.

[10] E. Deelman, G. Singh, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005. ISSN 1058-9244.

[11] K. Droegemeier, D. Gannon, et al. Service-oriented environments for dynamically interacting with mesoscale weather. *Computing in Science & Engineering*, 7(6):12–29, 2005.

[12] E. Elmroth, F. Hernández, et al. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. *Future Generation Computer Systems*, 26(2):245–256, 2010. ISSN 0167-739X.

[13] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *IFIP International Conference, Beijing, China*, 2005.

[14] J. Frey. Condor DAGMan: Handling inter-job dependencies, 2002.

[15] Y. Gil, V. Ratnakar, et al. Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In *Proceedings of the National Conference on Artificial Intelligence*, vol. 22, p. 1767. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[16] C. A. Goble and D. C. De Roure. myexperiment: social networking for workflow-using e-scientists. In *WORKS '07: Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pp. 1–2. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-715-5.

[17] A. Goderis, C. Brooks, et al. Heterogeneous composition of models of computation. *Future Generation Computer Systems*, 25(5):552–560, 2009.

[18] Y. Han, A. Sheth, et al. A taxonomy of adaptive workflow management. In *Workshop of the 1998 ACM Conference on Computer Supported Cooperative Work*. 1998.

[19] J. Hayes, E. Peyrovian, et al. Workflow interoperability standards for the internet. *Internet Computing, IEEE*, 4(3):37–45, 2002. ISSN 1089-7801.

[20] Z. Jinde. Study on Interoperability of Workflow Management Systems. *Journal of University of Electronic Science and Technology of China*, 2, 2002.

[21] P. Kacsuk and G. Sipos. Multi-grid, multi-user workflows in the P-GRADE grid portal. *Journal of Grid Computing*, 3(3):221–238, 2005. ISSN 1570-7873.

[22] G. Kandaswamy and D. Gannon. A Mechanism for Creating Scientific Application Services on Demand from Workflows. In *International Conference on Parallel Processing Workshops*, pp. 25–32. 2006.

[23] K. Klingenstein and D. Gannon. A workshop on scientific and scholarly workflow cyberinfrastructure: Improving interoperability, sustainability and platform convergence in scientific and scholarly workflow. Tech. rep., NSF and Mellon Foundation, 2007.

[24] S. Krishnan, L. Clementi, et al. Design and evaluation of opal2: A toolkit for scientific software as a service. In *Proceedings of the 2009 Congress on Services - I*, pp. 709–716. IEEE Computer Society, Washington, DC, USA, 2009. ISBN 978-0-7695-3708-5.

[25] S. Marru, S. Perera, et al. Reliable and Scalable Job Submission: LEAD Science Gateways Testing and Experiences with WS GRAM on TeraGrid Resources . *TeraGrid Conference*, June 2008.

[26] T. Oinn, M. Addis, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 2004. ISSN 1367-4803.

[27] S. Shirasuna. *A dynamic scientific workflow system for the Web services architecture*. Ph.D. thesis, Indiana University, 2007.

[28] C. Team. Dagman (directed acyclic graph manager). Http://www. cs. wisc. edu/condor/dagman.

[29] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *ACM Sigmod Record*, 34(3):44–49, 2005. ISSN 0163-5808.

[30] M. Zur Muehlen. A Framework for XML-based Workflow Interoperability–The AFRICA Project. In *Americas Conference on Information Systems*. Citeseer, 2000.