Task Decomposition for Adaptive Data Staging in Workflows for Distributed Environments

Onyeka Ezenwoye[1], Balaji Viswanathan[4], S. Masoud Sadjadi[2], Liana Fong[3], Gargi Dasgupta[4], and Selim Kalayci[2]
[1] South Dakota State University, Brookings, SD, USA, onyeka.ezenwoye@sdstate.edu
[2] Florida International University, Miami, FL, USA, sadjadi,skala001@cs.fiu.edu
[3] IBM Watson Research Center, Hawthorne, NY, USA, llfong@us.ibm.com
[4] IBM India Research Lab, New Delhi, India, bviswana,gdasgupt@in.ibm.com

***Abstract*—Scientific workflows are often composed by scientists that are not particularly familiar with performance and fault-tolerance issues of the underlying layer. The inherent nature of the infrastructure and environment for scientific workflow applications means that the movement of data comes with reliability challenges. Improving the reliablility scientific workflows in distributed environments, calls for the decoupling of data staging and computation activities, and each aspect needs to be addressed separately**

In this paper, we present an approach to managing scientific workflows that specifically provides constructs for reliable data staging. In our framework, data staging tasks are automatically separated from computation tasks in the definition of the workflow. High-level policies can be provided that allow for dynamic adaptation of the workflow to occur. Our approach permits the separate specification of the functional and non-functional requirements of the application and is dynamic enough to allow for the alteration of the workflow at runtime for optimization.

Keywords: Data Staging, Scientific Workflow, Distributed Systems.

I. Introduction

In a distributed computing system, components may reside in different physical locations. These components are often encapsulated as self-containted and internet-accessible software components or applications. Distributed applications are exposed as reusable components that can be dynamically discovered and integrated to create new applications. These new applications form aggregate (or composite) services. In this composite model, the composite application is an aggregation of tasks that are performed/executed by the integrated distributed applications, as illustrated in Figure 1.

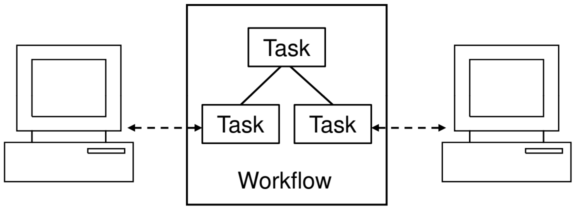

Fig. 1. Workflows aggregate tasks the get executed on the distributed computers

In the research community, these distributed applications and the ability aggregate them is allowing scientist to create complex scientific applications. To facilitate this integration are computing infrastructures such as the Grid [6] that allow for the harnessing of resources available on disparate distributed computing environments to create a parallel infrastructure that allows for applications to be processed in a distributed manner. These applications, which are geared towards scientific discovery tend to be compute and data intensive, requiring large amounts of data to be moved around the system. Since moving the application close to the data is not always practical due to insufficient computational resources at the storage site [8], data needs to be moved to the applications that need them and in some cases cleanup operations need to occur after application execution. The inherent nature of the infrastructure and environment for these applications means that the migration of data comes with certain challenges. The successful execution of applications is dependent of the availability of necessary data. For instance, workflow mapping techniques may produce workflows that are unable to execute due to the lack of the disk space necessary for the successful execution [10], requiring that data movement be scheduled and monitored. In fact, the mangement of data is essential through the entire lifecycle of the workflow from creation to execution, and result management [4]

Scientific workflows are often composed by scientists and as such are not particularly tuned for performance and fault-tolerance [3] This is becaue workflow languages permit the abstraction of language semantics at a level that is easy for domain specialist to use, thus focus is often placed on the functional aspects of the workflow. Also, since the eventual execution resources are not known during composition, optimizing the runtime of the overall workflow becomes a big issue [3] Allthough data is a key component in scientific workflow, a lot of emphasis is not placed on providing fault tolerance for tasks related to their data requirements. Data staging tasks are often embedded in computation-related tasks and relaibility efforts are then focused on the computation tasks even though data access presents the main bottleneck for data-intensive applications [8]

Improving the reliablility of data placement in distributed environments, calls for the decoupling of data movement and computation activities, each aspect needs to be addressed separately [9] Infrastructure for distributed scientific applications needs to consider data movement as part of the end-to-end performance of the system and care must be taken to make sure they complete successfully and without any need for human intervention [8]

In this paper, we present an approach to managing scientific workflows that specifically provides constructs for reliable data staging. In our framework, data staging tasks are automatically separated from computational tasks in the definition of the workflow. High-level policies are provided that allow for dynamic adaptation to occur. Recovery actions are applied separately for either data or computation-related tasks, for failures that could arise from software, network or storage system. Our approach permits the separate specification of the functional and non-functional requirements of the application and is dynamic enough to allow for the alteration of the workflow at runtime for optimization.

The rest of this paper is is structured as follows. In section II presents the architecture of our adaptive workflow manager that decomposed data staging and computation. Section III contains some related work. Finally, some concluding remarks are provided in Section IV.

II. Workflows in Grid Environments

In this section we present a brief overview of the overall architecture of our workflow management system for grid environments. As part of the Latin American Grid (LA Grid) [2], we have developed a distributed architecture that is comprised of two main middleware components: the workflow manager and the meta-scheduler. The LA Grid model is an end-to-end, layered architecture that is comprised of five main layers (see Figure 2): the Application Layer, which models the business logic of the a complex application in a workflow; the Workflow Management Layer, which enacts the business logic of the workflow and is responsible for maintaining concurrency and sequencing among tasks (or jobs) in the workflow; the Meta-Scheduling Layer, which is responsible for resource selection and job execution control; the Local Resource Management Layer, which is responsible for scheduling and executing individual jobs on the local resources; and the Resource Layer, which is comprised of the actual computing, storage, and networking resources.

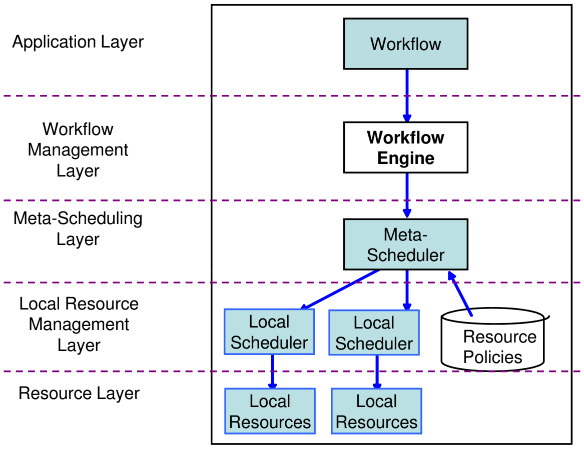

Fig. 2. A layered architecture for workflow execution in grid environments

To express the workflows themselves, we chose the Web Services Business Process Execution Language (BPEL) [5], which has emerged as the standard workflow language for orchestrating service-based applications. Several production-level software provide core BPEL engines. These engines are virtual machines that interpret and execute BPEL grammar. The grammar models the business logic of the workflow as a directed-graph, where the nodes represent tasks and the edges represent inter-task dependencies, data flow or flow control.

Currently, the BPEL specification does not contain the necessary semantics or support for defining jobs. Grid jobs require the richness and flexibility for specifying varied resource requirements and system environments. The Open Grid Forum job scheduling working group recommends the use of Job Submission Definition Language (JSDL) [1], for capturing a job's resource and environment requirements as well as data dependencies. In absence of unified modeling support, BPEL and JSDL are used to provide the combined modeling semantics for the workflow. This way individual workflow tasks are represented as JSDL jobs, embedded with BPEL constructs. This provides the necessary environment based on standardized technologies.

A. Data Staging

Many Grid jobs require input data, and in the absence of a shared file system, these datasets need to be staged in at the site of execution. Usually the data stag-in needs to be completed before the job can begin execution. In wokflows, the data requirement could be an input to the system or produced by the execution of a preceding job. In the latter case, a data-dependency is created in the flow between the producer and the consumer jobs of the data. Thus a typical data staging pattern in worklows comprises of a data stage-in from either producer jobs or from defined inputs, followed by a job submission pattern. In some cases, a data stage-out is specified to perform data cleanup operations after execution. There may be several such data staging activities, which could occur sequentially or in parallel.

Once the data staging of all dependencies are satisfied, a job can be submitted for execution. Typically, data staging activities are embedded with the specification of the the computation task in the JSDL document, as illustrated in Figure 3 This complex JSDL is then submitted as one job submission request by the workflow. Within this framework, it becomes difficult to isolate the source of failures. For instance, faults generated by data stage-in can get propagated to job execution.

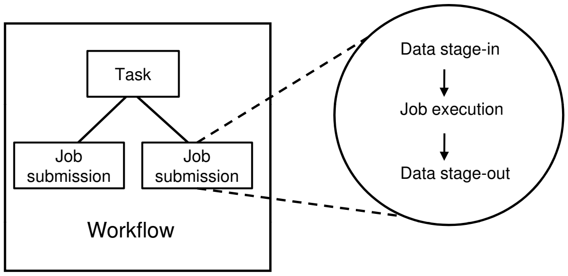

Fig. 3. A job submission task with embedded data staging activities.

B. Decoupling Data Staging

In this section, we present our approach to providing an adaptive workflow execution. Our approche takes into consideration the need provide adaptive data staging as part of the end-to-end performance of the workflow, by automatically decoupling the specification of data staging and computation. Not decoupling data staging and computation affects capability of system to provide fault-tolerance and adapt to environment and user prefrences. Data staging jobs and computational jobs need to be differentiated from each other within the system. Figure 4 illustrates the architecture of our adaptive workflow manager.

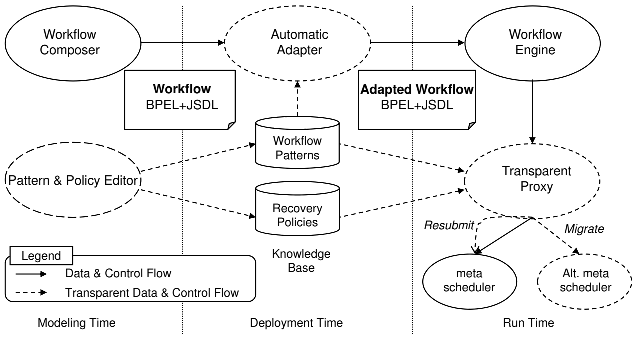

Fig. 4. The architecture of our adaptive workflow manager

In the left side of Figure 4, a domain expert will use the Workflow Composer to specify the business logic of the application using BPEL+JSDL. The domain expert should only be concerned about the business logic of the application and not about handling faults and exceptions. The job descriptions (in JSDL) are treated as XML complex types, which in turn are used as the parameters to some Invoke constructs in BPEL. It is within this JSDL definitions that data staging and computation tasks are encapsulated. During deployment time, the resulting workflow is passed to the Automatic Adapter, which automatically generates a functionally equivalent workflow. It is during this adaptation phase that the complex JSDL definitions that data staging and computation tasks are decomposed in primitive JSDL definition. Separate definitions and invocations are defined in the workflow for data stage-in, computation, and data stage-out. The invocation messages are extended with context information so that correlations can be made between the decomposed tasks. The context information is also needed for the Proxy to monitor the interaction between the workflow manager and the meta-schedulers.

The automatic adapter has an algorithm that identifies the known workflow patterns (e.g. job submission and data staging) within the workflow. The most updated workflow patterns are stored in the Knowledge Base. New workflow patterns can be added to the knowledge base using the Pattern and Policy Editor. The generated workflow, called adapted workflow, would not include constructs to handle faults at run time. Instead workflow behavior is modified at run time through the Transparent Proxy. At run time, the workflow will be executed by the Workflow Engine. The workflow engine can be any standard BPEL engine, as we did not extend BPEL in our work. During the automatic adaptation of the workflow, all the calls originally targeted for the local Meta-scheduler are redirected to the Transparent Proxy [7]. Therefore, the Transparent Proxy will intercept all the calls to the Meta-scheduler.

The Proxy will appear as a Meta-scheduler to the workflow process, and as a workflow process to the Meta-scheduler; hence, the name transparent. Its main responsibility includes submission of the jobs to the local Meta-scheduler and notifying the workflow process of the job status when it receives job status updates from the Meta-scheduler. In addition, it implements a pattern-matching algorithm that monitors the behavior of the intercepted calls and provides fault-tolerant behavior when faults occur. The algorithm is based on the Recovery Policies, the context information embedded in the adapted workflow, the Workflow Patterns, and their corresponding Fault-Tolerant Patterns. For example, following the recovery policies governing the current faulty situation, the Transparent Proxy may resubmit the job (data staging or computation) to the same Meta-scheduler or migrate it to another Meta-scheduler.

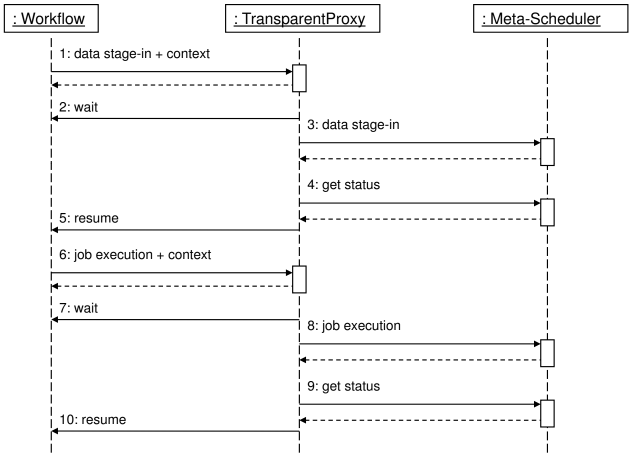

Fig. 5. Sequence diagram showing the interaction between the workflow engine, transparent proxy and meta-scheduler

Figure 5 shows the interaction between the interaction between the workflow engine, transparent proxy and meta-scheduler. Some messages have been simplified or removed for the purpose of brevity. As depicted in the diagram, data stage-in and job execution submission are decomposed and separated (data stage-out is not shown). Data Stage-in jobs are submitted first to the proxy with some context information that it needs to correlate related job execution and data stage-out submissions. The workflow is made to wait while the proxy attempts to execute the data stage-in task. The proxy will apply recovery actions (such as retry or migrate) based on specified high-level policies. Upon successful data staging, the workflow is allowed to proceed and further tasks can be submitted.

III. Related Work

The work by Chervenak [3] is concerned with data placement policies that distribute data in ways that are advantageous for application execution, for instance, by placing data sets so that they may be staged into or out of computations efficiently or by replicating them for improved performance and reliability. Their work centers on prestaging data using the Data Replication Service versus using the native data stage-in mechanisms of the Pegasus workflow management system. A policy-driven data placement service is responsible for replicating and distributing data items in conformance with policies or preferences. This work differs from our because it applies data management techniques at the local resource management layer (see Figure 4), while our work focuses on the workflow management layer.

Kosar [8] presents a data placement subsystem that allows for data for distributed computing systems to be queued, scheduled, monitored, managed, and checkpointed. Their framework includes a specialized scheduler for data placement, a high level planner aware of data placement jobs, a resource broker/policy enforcer and optimization tools. Data placement jobs are represented in a different way than computational jobs in the job specification language so that the high level planners can differentiate these two classes of jobs. The system can perform reliable data placement, and recover from failures without any human intervention. This work does is not dynamic since it requires that the workflow definition be modified and redeployed in order for any adaptation to occur. In comparison, our approach is automatic and includes context information for better fault tolerance. Also by focusing on the workflow management layer, we assume no control over data and job scheduling.

Ranganathan's [9] framework allows for data movement operations may be tightly bound to job scheduling decisions or, performed by a decoupled , asynchronous process on the basis of observed data access patterns and load. A scheduling framework within which a wide variety of scheduling algorithms can be used. They assume a multi-user and multi-site model. At each site, there are 3 components: an External scheduler (ES) that dertmines where to send jobs submitted to that site; a local scheduler (LS) that determines the order in which jobs are executed at that particular site; and a Dataset scheduler (DS) that detemines when to replicate data/and or delete local files. This work differs from ours because it considers data management issues at the local resource management layer, while our work focuses on the workflow management layer.

Singh [10] focuses on optimizing disk usage and scheduling large-scale scientific workflows onto distributed resources. Their approach is minimize the amount of space a workflow requires during execution by removing data files at runtime when they are no longer needed. To achieve this, workflows are restructured to reduce the overall data footprint of the workflow. Their algorithms adds a cleanup job (data stage-out) for a data file when that file is no longer required by other tasks in the workflow. Similar to our approach, their workflow adaptation algorithm is applied after the executable workflow has been created but before the workflow is executed. However, the issue of fault-tolerance is not addressed and no data cleanup if a compute task fails.

IV. Conclusion

In this paper, we present an approach to managing scientific workflows that specifically provides constructs for reliable data staging. In our framework, data staging tasks are automatically separated from computational tasks in the definition of the workflow. High-level policies can be provided that allow for dynamic adaptation of the workflow to occur. Recovery actions are applied separately for either data or computation-related tasks, for failures that could arise from software, network or storage system. Our approach permits the separate specification of the functional and non-functional requirements of the application and is dynamic enough to allow for the alteration of the workflow at runtime for optimization.

Acknowledgement: This work was supported in part by IBM, the National Science Foundation (grants OISE-0730065, OCI-0636031, HRD-0833093, and IIP-0829576). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the NSF and IBM.

References

[1] A. Anjomshoaa, A. Anjomshoaa, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. *Job Submission Description Language(JSDL) Version 1.0*, November 2005.

[2] R. Badia, G. Dasgupta, O. Ezenwoye, L. Fong, H. Ho, Y. Liu, S. Luis, A. Praino, J.-P. Prost, A. Radwan, S. M. Sadjadi, S. Shivaji, B. Viswanathan, P. Welsh, and A. Younis. Innovative grid technologies applied to bioinformatics and hurricane mitigation. In *High Performance Computing and Grids in Action*. IOC Press.

[3] A. Chervenak, E. Deelman, M. Livny, M.-H. Su, R. Schuler, S. Bharathi, G. Mehta, and K. Vahi. Data placement for scientific applications in distributed environments. *IEEE/ACM International Workshop on Grid Computing*, 0:267–274, 2007.

[4] E. Deelman and A. Chervenak. Data management challenges of data-intensive scientific workflows. In *Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 687–692, Washington, DC, USA, 2008. IEEE Computer Society.

[5] O. Ezenwoye and S. M. Sadjadi. Composing aggregate web services in BPEL. In *Proceedings of The 44th ACM Southeast Conference*, Melbourne, Florida, March 2006.

[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[7] S. Kalayci, O. Ezenwoye, B. Viswanathan, G. Dasgupta, S. M. Sadjadi, and L. Fong. Design and implementation of a fault tolerant job flow manager using job flow patterns and recovery policies. In *Proceedings of the 6th International Conference on Service Oriented Computing (ICSOC'08)*, volume 5364/2008, pages 54–69, Sydney, Australia, December 2008. Springer Berlin / Heidelberg.

[8] T. Kosar and M. Livny. A framework for reliable and efficient data placement in distributed computing systems. *Journal of Parallel and Distributed Computing*, 65(10):1146–1157, 2005.

[9] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. volume 0, page 352, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[10] G. Singh, K. Vahi, A. Ramakrishnan, G. Mehta, E. Deelman, H. Zhao, R. Sakellariou, K. Blackburn, D. Brown, S. Fairhurst, D. Meyers, B. Berriman, J. Good, and D. Katz. Optimizing workflow data footprint. *Scientific Programming*, 15(4):249–268, 2007.