A High-Level Distributed Execution Framework for Scientific Workflows

Jianwu Wang[1], Ilkay Altintas[1], Chad Berkley[2], Lucas Gilbert[1], Matthew B. Jones[2]

[1] *San Diego Supercomputer Center, UCSD, U.S.A.*
{jianwu, altintas, iktome}@sdsc.edu

[2] *National Center for Ecological Analysis and Synthesis, UCSB, U.S.A.*
{berkley, jones}@nceas.ucsb.edu

Abstract

Domain scientists synthesize different data and computing resources to solve their scientific problems. Making use of distributed execution within scientific workflows is a growing and promising way to achieve better execution performance and efficiency. This paper presents a high-level distributed execution framework, which is designed based on the distributed execution requirements identified within the Kepler community. It also discusses mechanisms to make the presented distributed execution framework easy-to-use, comprehensive, adaptable, extensible and efficient.

1. Introduction

Scientific workflow management systems, e.g., Taverna [1], Triana [2], Pegasus [3], Kepler [4], ASKALON [7] and SWIFT [12], have demonstrated their ability to help domain scientists solve scientific problems by synthesizing different data and computing resources. Scientific workflows can operate at different levels of granularity, from low-level workflows that explicitly move data around, start and monitor remote jobs, etc. to high-level "conceptual workflows" that interlink complex, domain specific data analysis steps. Distributed execution and Grid workflows can be seen as a type of scientific workflows. Most workflow systems centralize execution [5], which often causes a performance bottleneck. We summarize requirements within the Kepler community and propose our distributed execution framework to take advantage of abundant distributed computing resources to achieve better execution performance and efficiency. Based on community feedback, our goals for the Kepler distributed execution framework include the ability to easily form ad-hoc networks of cooperating Kepler instances. Each cooperating Kepler network can impose access constraints and allows Kepler models or sub-models to be run on participating instances. Once a Kepler cooperating network has been created, it can configure one or more subcomponents of a workflow to be distributed across nodes of the newly constructed network. The major contribution of this paper is demonstrating a distributed scientific workflow approach that combines an intuitive user interface, collaborative features, and capabilities for distribution of workflow tasks and the workflows themselves in a single framework.

In Section 2, we discuss the background of scientific workflow distributed execution. Sections 3 and 4 describe the conceptual architecture and framework. We demonstrate a case study in Section 5 to show how the framework works. Finally, we conclude and explain future work in Section 6.

2. Background

Our work is based on the following aspects: structure of scientific workflow specifications, typical distributed execution requirements as specified by scientists, and prior work in distributed execution.

2.1. Scientific Workflow Specification Structure

There are several different formats for representing scientific workflows [14, 15, 16], but they generally are graph descriptions that can be used to represent three types of components: tasks, data and control dependencies [5]. For example, in Figure 1 the tasks T2 and T3 will be executed under different conditions. Additionally, T4 needs to get data from either T2 or T3 before its execution. Since our framework incorporates

978-0-7695-3535-7/08 $25.00 © 2008 IEEE
DOI 10.1109/eScience.2008.166

Table 1. Requirements for distributed execution in the Kepler user community.

Requirement	Explanation	Approach
Execute Tasks on Remote Nodes	Some tasks cannot be executed locally for the following reasons: 1) insufficient local computing cycles for desired task; 2) scientific codes in, e.g., C and Fortran, are not ported to run on the local computer; 3) third party software, e.g., Matlab, may not be installed on the local computer; 4) the user does not have authority to run a third-party application.	Distribute to other computers that can execute the required tasks.
Distributed Node Discovery	Users know they can benefit from distributed execution, yet they do not know where to find and choose proper distributed nodes.	Automatic distributed node discovery and selection.
Peer-to-Peer Data Transfer	Centralized transfer of large datasets is one of the main bottlenecks for execution efficiency.	Transfer data between source and destination nodes directly, or execute the task on the node containing the data.
Provenance of Distributed Execution	Provenance can help users to track execution information in the future [6], which should also be supported in distributed execution environments.	Provide proper locations to store provenance information and ways to track it easily in distributed environments.
Distributed Monitoring	Users need to know the current execution status, e.g., which tasks are executing on which computers with which data.	Support graphical monitoring and user interaction for more detailed information.
Transparent Implementation	Distributed execution should be usable with little or no knowledge of grid computing techniques. Most of the Kepler user community are unfamiliar with existing grid computing frameworks, e.g., Globus, and requiring knowledge of these types of systems effectively prevents adoption of distributed computing.	Automatically choose proper implementation techniques according to real situations without user interaction with arcane configuration languages.
Reuse Existing Workflows	Workflows that can be executed centrally should be distributed with minimal changes to the workflow specifications.	Decouple workflows' execution from their specifications.
Failure Recovery	Monitor, diagnose and recover from failures during distributed execution.	Find failure reasons, re-allocate distributed nodes, and re-execute failed tasks.

this basic workflow specification structure, it can be implemented for different scientific workflow engines.

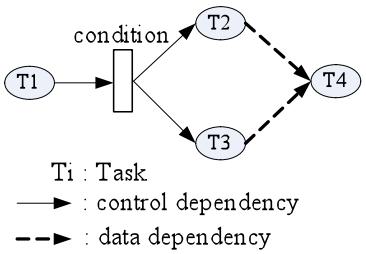

Figure 1. An example workflow composed of tasks and dependencies.

2.2. Typical Requirements for Distributed Execution using Scientific Workflows

The Kepler project aims to produce an open-source scientific workflow system that allows scientists to design and efficiently execute scientific workflows. Since 2003, Kepler has been used as a workflow system within over 20 diverse projects and multiple disciplines. Within the Kepler community, we have identified various requirements for scientific workflows related to distributed execution (Table 1).

2.3. Related Work

Several scientific workflow systems support distributed execution. Triana [2] provides two kinds of distributed execution topologies: parallel computation for a single task and pipelined connectivity with direct data transfer between different tasks. Pegasus [3] supports resource allocation and data provenance mechanisms in Grid-based distributed environments. ASKALON [7] contains a service repository for service broker and data repository for data sharing.

After evaluating these systems, our high-level distributed execution framework focuses on the following features:

- *Easy-to-use:* the demands on users, especially scientists, should be as simple as possible.
- *Comprehensive:* satisfies all the requirements in Table 1.
- *Adaptable:* works well with different scientific workflow specifications and distributed environments.
- *Extensible:* new processing components can be easily added to the framework to meet new requirements.

- *Efficient:* minimize the overhead when using distributed execution.

3. Conceptual Architecture

In this distributed framework, each computing node runs an instance of the workflow execution engine and is assigned one or more roles. Workflow execution is initiated by a *Master* node that performs overall coordination of an arbitrary number of *Slave* nodes that execute workflow tasks. Additionally, a *Registration Center* and a *Provenance Manager* broker Slave execution capability and data generated during workflow execution respectively.

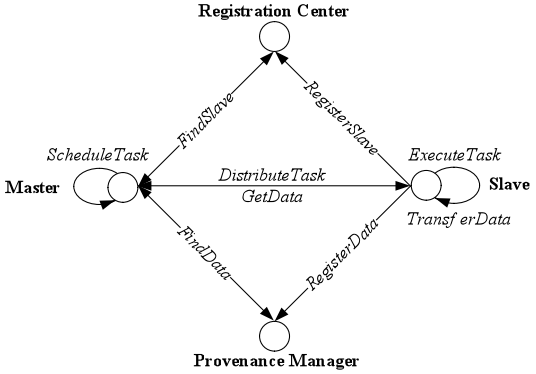

Figure 2. Distributed execution conceptual architecture.

Master: The Master interacts with end users and directs workflow execution. The Master calculates the proper workflow task schedule and distributes tasks to Slaves for execution using the Slave information from the Registration Center. In addition, the Master retrieves concrete data content from Slaves using data register information found from the Provenance Manager.

Slave: Slaves are computing nodes that register their execution capability to the Registration Center for the Master to query. The Slave also registers provenance information at the Provenance Manager for the Master to query. (see Section 4)

Registration Center: The Registration Center manages the registered Slave information. Masters can query the Registration Center to discover information about current available Slaves and their states.

Provenance Manager: The Provenance Manager is used to register and query execution information, e.g., task state, and data that was generated during workflow execution. Masters can monitor current execution status and track the provenance information. Provenance information is also useful for failure diagnosis and partial workflow re-execution [8].

The requirements listed in Section 2.2 are met using this framework. Adaptations can be implemented for different distributed environments and scientific workflow specifications. In actual applications, there are usually several Masters, several Slaves, one Registration Center, and one Provenance Manager within a logical domain. Each node may fill multiple roles.

The main interfaces of each role and the typical interaction sequences between them are illustrated by sequence diagram in Figure 3. Figure 3 shows the usage of a Registration Center and a Provenance Manager by different Masters and Slaves for the decoupled architecture.

4. Working Mechanisms

The conceptual architecture explained in Section 3 can be implemented in different ways. In this section, we will discuss some important mechanisms that make the implementation easy-to-use, comprehensive, adaptable, extensible and efficient.

Decoupling of the Workflow Specification from the Execution Model. Through decoupling the specification of the workflow from its execution model, existing workflow specifications can be reused when changing from centralized execution to distributed execution. In Kepler [4], the workflow specification is loosely coupled with the execution model, i.e., Director. The Director specifies the model of computation under which the workflow will run and the user can easily change execution model by replacing the Directors using the graphical user interface for designing workflows. We apply this principle to distributed computing. Ideally, a user is able to distribute workflow execution simply by replacing the Director of her existing workflow to the corresponding distributed Director. There are experimental implementations of new Directors for distributed execution, such as a massively parallel parameter sweep using dynamic dataflow model [13] or automated distributed simulation using synchronous dataflow model [9]. More experiments are needed to test the fitness and performance of each Director in our proposed distributed execution architecture.

Peer-to-Peer Data Transfer. In order to improve data transfer efficiency during distributed execution, we employ a pipeline mechanism to realize peer-to-peer data transfer, which is similar to [2] and [3]. A corresponding pipeline is established for each data dependency when the workflow is initiated, and will transfer from source Slave to destination Slave(s) directly. In distributed environments, each pipeline needs to have global access, e.g., through a URI or local proxy, to establish data connections among distributed Slaves.

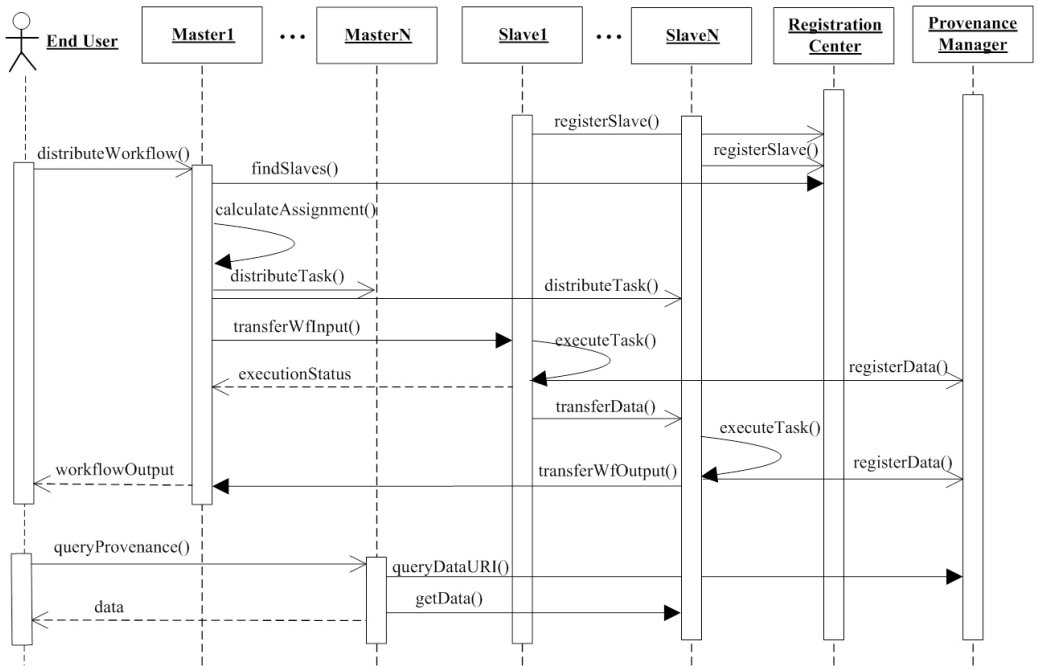

Figure 3. Typical interaction sequence during distributed execution of a scientific workflow.

Transparent Implementation. By making technologies, such as JINI, Grid Services and Web Services, transparent to users, end users with little or no computer knowledge can also distribute the execution of their workflows. For example, Taylor et al. [2] describe support for different infrastructure bindings through general interfaces that lets users choose which distributed technologies to use. We plan to improve this capability by defining technology selection rules and detecting the context of real situations. In our high-level framework, features for transparent distribution, execution and data transfer need to be enabled without interacting with users. In addition, ease of deployment is critical to adoption. Most scientific users in our community have no interest or background in setting up distributed computing systems on their compute nodes. Our proposed solution eliminates the need to set up a separate grid-computing system by enabling each node running workflow instance to act as an execution endpoint in either the Master or Slave role. This approach significantly eases the deployment burden that typically inhibits the use of grid computing frameworks.

Capability-Based Slave Registration. Detailed information (called the *Slave Execution Capability*) of each registered Slave is maintained at the Registration Center. Similar to the site catalog in [3], the Registration Center stores hardware, software and other information on each Slave. This information is serialized in XML to make it extensible. The meta model for Slave Execution Capability is shown in Figure 4. The main components are: *TaskSet*: describes which tasks are registered to be executable on the Slave; *Database*: the domain specific databases stored on the Slave; *Hardware*, *RealTimeStatus*, *Security*, and *Stereotypes*, i.e., extension mechanisms, for adding new elements. Depending on the characteristics of the distributed environment, some elements may be not used and new ones may be extended for specific requirements.

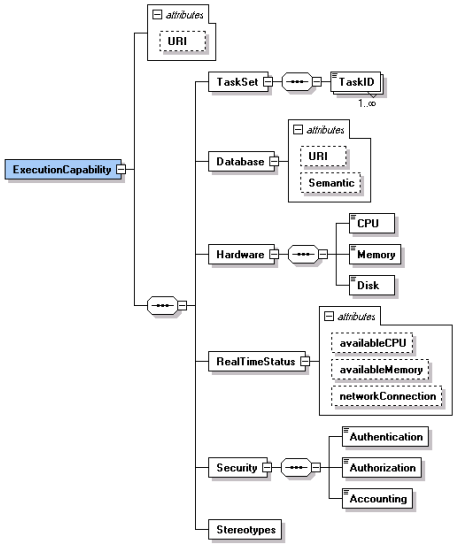

Figure 4. Slave Execution Capability metamodel.

Automatic Constraint-Based Task Scheduling. To achieve optimal task scheduling from numerous possible solutions, the Master needs to match user requirements with Slave execution capabilities. Besides the basic functional requirements, e.g., the task scheduling solution must be able to complete the workflow execution, users may have additional non-functional constraints, such as throughput, and time to completion. Therefore a mechanism matching user requirements and Slave capabilities need to provide a solution that not only meets the functional requirements but also the non-functional constraints. Many task-scheduling algorithms have been studied in Grid-based systems [10], and can be made useful in our framework. However, the assumption in [10] that each task has an estimated run-time is not applicable in the context of our framework as run-times of some tasks vary with different input configuration, e.g., Web Service actor in Kepler. New algorithms need to be proposed to take the task's input and configuration values into account.

Broker based Provenance Management. Broker-based provenance management is employed to realize the tradeoff between functionality and efficiency of distributed provenance. In distributed environments, there are two typical ways to store the execution information for data provenance: centralized and decentralized. It is inefficient to store the data content of all distributed nodes in one centralized center because data in e-Science may be huge. It is efficient to separate data storage in each distributed node locally, but this makes it difficult to query and integrate this data in the future. For example, a task may be distributed from different Masters for different executions, and distributed to different Slaves each time. It is hard to collect the overall execution information of the workflow in this situation. So unlike the centralized provenance catalog in [11], our framework utilizes a separate Provenance Manager to broker the provenance data. The basic information in the Provenance Manager consists of the description and the endpoint of each data item. After finding the needed data endpoint at the Provenance Manager, the Master can get data content from the corresponding Slaves. In this way, the burden for data transfer is reduced compared to the centralized provenance system. This also makes it easier to do future provenance tracking.

5. Case Study

In this section, we will use a case study which is simple yet common in the Kepler community in order to apply our distributed framework. Three domain scientists collaborate and construct a workflow with tasks in their separate sub-domains. Individual tasks of the workflow are executable on some of the scientists' personal computers, yet the whole workflow cannot be executed on any one computer. This is an example of the "Execute Tasks on Remote Nodes" requirement from Table 1. The scientists hope to connect their computers to execute the workflow and track the provenance information for this execution. Yet they are not computer professionals and have no experience with distributed computing technologies.

From the users' perspective, our distributed framework is an analysis and modeling tool that provides functionalities to support distributed execution (such as starting and registering their nodes as Slaves). Using the framework, domain scientists can design and configure an ad-hoc network of nodes where they assign each computer to serve one or more of the roles defined in Section 3. From the Master node's graphical user interface, they can specify the workflow to be distributed and trigger its execution. With the Slave information registered at the Registration Center, the Master automatically finds a proper task scheduling solution and distributes the workflow to the Slaves. Tasks are executed at the corresponding Slaves in parallel or sequentially based on the workflow specification. Data is transferred directly from source Slaves to destination Slaves according to the data dependency graph. Data generated during the execution is registered with the Provenance Manager. Any of the scientists can track the provenance information from the workflow GUI on his or her own computer by finding data registration information from the Provenance Manager and then retrieving data content directly from the Slave that stores the data. An example of the distributed framework deployment for this case and the data transfer among them is shown in Figure 5.

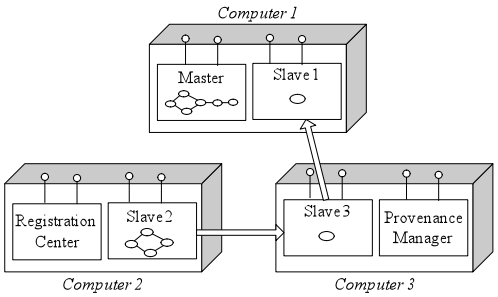

Figure 5. An example of distributed execution framework deployment.

6. Conclusion and Future Work

Distributed workflow execution is a way to achieve better performance and efficiency when conducting computational experiments. Based on the requirements from the Kepler community, we present a high-level distributed execution framework and discuss its main working mechanisms. We are extending the current implementation of distributed computing features in Kepler according to the presented framework. Although this framework focuses on features for easy-to-use, comprehensive, adaptable, extensible and efficient distributed execution, the biggest focus of our work is on usability of the platform in terms of adoption in our community. It builds upon the existing work and solutions for some of the desired features including error handling, scheduling algorithms and minimizing overhead.

The paper is the first description of our discussions and design on the subject and serves as a reference for future work. We will refine the design details for key factors, e.g., task scheduling optimization and techniques for implementation transparency. The implementation in Kepler based on this initial design and evaluation of design decisions is on-going work.

7. Acknowledgements

The authors would like to thank the rest of the Kepler team for their collaboration, especially to the Kepler/CORE distributed execution interest group for the discussion on requirements for a distributed execution framework, This work was supported by NSF SDCI Award OCI-0722079 for Kepler/CORE, NSF ITR Award No. 0225676 for SEEK and NSF CEO:P Award No. DBI 0619060 for REAP.

8. References

[1] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver and K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics, 20(17), pp. 3045-3054, Oxford University Press, London, UK, 2004.

[2] I. Taylor, M. Shields, I. Wang, and A. Harrison. The Triana Workflow Environment: Architecture and Applications. In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, Workflows for e-Science, pp. 320-339. Springer, New York, Secaucus, NJ, USA, 2007.

[3] E. Deelman, G. Mehta, G. Singh, M. Su, and K. Vahi. Pegasus: Mapping Large-Scale Workflows to Distributed Resources. In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, Workflows for e-Science, pp 376-394. Springer, New York, Secaucus, NJ, USA, 2007.

[4] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, Y. Zhao. Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience, 18 (10), pp. 1039-1065. 2005.

[5] J. Yu and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing, 2006 (3), pp.171-200.

[6] S. Davidson, S. Cohen-Boulakia, A. Eyal, B. Ludäscher, T. McPhillips, S. Bowers, M. Anand and J. Freire. Provenance in scientific workflow systems. IEEE Data Engineering Bulletin, 30(4), pp. 44-50, 2007.

[7] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, Jr, and H. Truong. ASKALON: a tool set for cluster and Grid computing. Concurrency and Computation: Practice and Experience, 17(2-4), pp. 143-169, Wiley InterScience, 2005.

[8] D. Crawl and I. Altintas. A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows. In proceedings of 2nd Intl' Provenance and Annotation Workshop (IPAW 2008). June, 2008.

[9] D. Cuadrado. Automated Distribution Simulation in Ptolemy II. PhD thesis, Aalborg University, April, 2008. Available at http://chess.eecs.berkeley.edu/pubs/412.html.

[10] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal and K. Kennedy. Task Scheduling Strategies for Workflow-based Applications in Grids. IEEE International Symposium on Cluster Computing and Grid (CCGrid), 2005. pp. 759- 767, Vol. 2.

[11] E. Deelman, Y. Gil, Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges, e-science, pp.144-149, Second IEEE International Conference on e-Science and Grid Computing (e-Science'06), 2006.

[12] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, M. Wilde. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. 2007 IEEE Congress on Services (Services 2007), July 2007, pp. 199-206.

[13] D. Abramson, C. Enticott and I. Altinas. Nimrod/K: Towards Massively Parallel Dynamic Grid Workflows. To appear in Supercomputing 2008 (SC2008).

[14] C. Brooks, E.A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, H. Zheng (eds.), Chapter 7: MoML, Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II), EECS Department, University of California, Berkeley, UCB/EECS-2008-28, April 1, 2008.

[15] Scufl Language, Taverna 1.7.1 Manual, http://www.mygrid.org.uk/usermanual1.7/.

[16] Virtual Data Language Reference Manual, http://www.ci.uchicago.edu/swift/guides/languagespec-0.6.php.