Flexible Grid Workflows Using TÆMS

James Atlas, Keith Decker, Martin Swany
Computer and Information Sciences
University of Delaware
Newark, DE 19716
atlas, decker, swany@cis.udel.edu

Abstract

Grid computing tasks are often broken down into multiple subtasks and connected using a directed acyclic graph (DAG) to form a grid workflow. If we assume a dynamic market-based environment with multiple virtual organizations competing to provide grid computing resources, it becomes apparent that a static mapping of these resources to the workflow will produce suboptimal results. Furthermore, the authors argue that a rigid workflow model will also produce suboptimal results within a dynamic environment.

The multi-agent systems community addresses the problem of executing high-level tasks through a series of coordination mechanisms. If we view grid computing service providers and consumers as agents, we can view the problem of executing a grid computing task as coordination between agents. The position taken in this paper is that TÆMS, a domain independent model of the problem solving activities of an intelligent agent, can provide a flexible workflow model that incorporates the notion of quality for grid computing tasks. This flexible workflow approach improves the capabilities of workflow execution engines to optimize a workflow.

Introduction

Directed acyclic graphs (DAG) can be used to represent the general workflow of a grid task. By representing different subtasks as nodes on the graph, the connecting edges can represent the control flow. Tasks can be executed in sequence or in parallel, and synchronization can be represented by connecting multiple endpoints to the same node. A rigid workflow model can be thought of as one in which one of two conditions holds: all nodes in the DAG must be visited to complete the task, or all decisions about which nodes to visit are determined by parameters local to the problem (results from previous tasks or static configuration). The second condition would represent a typical conditional branch. A flexible workflow model allows workflow engines and compilers to modify or selectively execute the underlying workflow to accomplish the task.

The process of designing a DAG for a computational task is usually done through a visual modeling tool or through a specific programming language. The GridAnt system uses the Apache tool, Ant, to provide a client oriented DAG workflow solution (Amin & von Laszewski 2003). Other proprietary solutions have used scripting languages to represent the flow (UNICORE 2004), (Con 2005). The Globus community has proposed a standard Grid Services Flow Language (GSFL) (Krishnan, Wagstrom, & von Laszewski 2002). This standard was based on the Web Services Flow Language (WSFL), but provided some specific enhancements for grid applications, namely peer-to-peer communication and on-demand services. The Global Grid Forum community has proposed an XML-based grid workflow vocabulary (Bivens 2001). The web services community, including industry giants IBM and Microsoft, has developed and endorsed the Business Process Execution Language for Web Services (BPEL4WS) (OASIS 2003). This specification merges and supercedes the earlier WSFL and XLang specifications (and will soon be changed to Web Services Business Process Execution Language, WSBPEL, in the upcoming 2.0 specification). While not aimed directly at grid applications, the BPEL4WS standard supports many distributed application constructs that prove useful, including sequential and parallel flow, fault tolerance and recovery, and controlled flow branching.

While some these models have sophisticated implementations, most limit the ability of the execution engine to negotiate acceptable quality of service for the task initiator. We could define the quality of service as a measure of task accuracy, but it is a more general concept. Intuitively, the higher the quality of service, the better the workflow result. For example, the quality of a large scale simulation could reflect how closely the simulation approximates the actual phenomenon. We assume that the engine is able to negotiate with potential service providers for cost and duration of tasks, but even with economic negotiation models such as Contract Net and scheduling support for service level agreements (loose reservations) the execution engine may not be able to reach the optimal quality of service. Given perfect information the engine might reach an optimal mapping of subtasks to resources, but it is limited in its ability to reason about the list of subtasks that it must execute.

Why might an execution engine want to reason about the subtasks? As a simple illustration, imagine two paths within the workflow that both enable the overall task to complete.

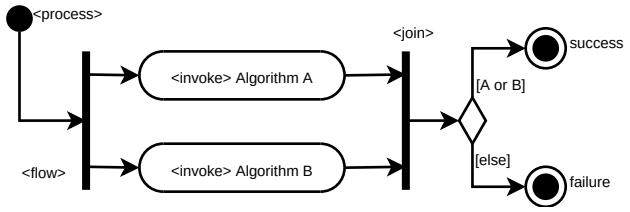

Figure 1: BPEL4WS representation of the classic OR

Each path represents a different algorithm, but each obtain the same results for a data set. Algorithm A uses twice as many CPU cycles as algorithm B, and algorithm B uses twice as much memory as algorithm A. The execution engine gains nothing by executing both paths, but must execute at least one to finish the overall task. This could be represented as a classic OR condition in the workflow. BPEL4WS is the only current workflow model that can handle this simple condition, as shown in Figure 1.

We can extend the idea of the classic OR to a non-boolean OR in which we consider the quality of service. In the classic sense, success quality could map to a value of 1 and failure quality to 0. By extension, the range of values between 0 and 1 would represent varying levels of quality. For example, suppose the previous algorithm A produced a quality of 0.2 and algorithm B produced a quality of 0.5. If both algorithms were executed, the non-boolean OR would obtain the resulting quality of 0.5. If only algorithm A was executed the non-boolean OR would obtain the resulting quality of 0.2. If the workflow engine needed to produce a quality of 0.1 then it is free to choose algorithm A or B. If it needed a quality of 0.4 then algorithm B is the only option.

Under current workflow languages, it is not possible to represent this situation. BPEL4WS provides flexibility with the classic OR case that enables the workflow engine to selectively execute tasks. However, it does not provide a representation of the task quality. Since the task quality is determined by the designer of the workflow, this limits the workflow engine's ability to provide the best quality result. Furthermore, there exist more complex task relationships that affect quality than the simple non-boolean OR. Current workflow languages do not enable workflow engines to handle these situations.

Approach

The problems discussed so far can be characterized by a lack of expressiveness and flexibility in current workflow models. One reason for this shortcoming is the domain from which these flow languages evolved. The domain of business to business (B2B) interactions has several properties that can be seen in the design of the workflow languages. The first property is runtime length. In most B2B transactions the runtime is very short, often on the order of seconds. A short runtime makes it disadvantageous to negotiate and re-plan a workflow because of the computation time involved. The second property is the rigidness of B2B structure itself. Predictable behavior is a tenet of financial and other mission-critical applications. These properties are still present in certain grid applications, but we argue that in general they are different. Grid computing emphasizes large scale tasks that most likely have runtimes on the order of hours or longer. It also has a notion of quality that is not present in typical B2B transactions. A quality level of 99% may be acceptable for a large graphics rendering application, whereas most businesses would frown on a quality level of 99% for a money transfer. These somewhat subtle differences in domains form the basis for a flexible workflow model.

In particular, a more robust model of task relationships will be necessary to make the grid workflow flexible enough to handle these situations. These task relationships have been studied in great detail in the multi-agent systems community. Multi-agent systems often operate in a satisficing mode in which the solution may not be 100% of the quality level, but instead meets some established minimum quality. In addition to the OR relationship explored earlier, other task relationships have emerged that lack representation in current workflow models. In the next section we propose adopting the TÆMS model for these task relations, and explore how this could be implemented for a grid computing task.

TÆMS models the task relations involved in problem-solving activities of intelligent agent systems (Horling *et al.* 1998). A task structure is represented as a graph that is hierarchically decomposed into subtasks. While allowing for recursivity in the graph, for simplicity we will refer to this task structure as a tree without much loss in expressiveness. This tree contains sequences of subtasks, or sub-goals, and methods. Methods represent leaf nodes of the tree that the model has not decomposed (without loss of generality we will refer to both subtasks and methods as tasks). TÆMS differentiates between two types of node relationships: parent/child and node-to-node (peer-to-peer). The structure for a parent/child connection is a quality accumulation function, and for a node-to-node connection is an interrelationship.

The first structure for connections, Quality accumulation functions (QAF), determines how the quality of child tasks combine to produce the quality of the parent task. Many functions have been developed for use within the model, but we will focus on three of them. They are min, max, and sum. From the TÆMS white-paper:

> A min QAF is functionally equivalent to an AND operator. The quality of the supertask is equal to the minimum quality of its subtasks.
>
> A max QAF is functionally equivalent to an OR operator. It says that the quality of the supertask is equal to the maximum quality of any one of its subtasks.
>
> The sum QAF says that the quality of the supertask is equal to the sum of the qualities of its subtasks, regardless of order or which methods are actually invoked.

In our earlier example of the non-boolean OR, the two paths could have been represented as child nodes to a parent using the max QAF, as shown in Figure 2. The execution engine could then reason about which child node to execute to produce the required quality at the parent node. A min QAF

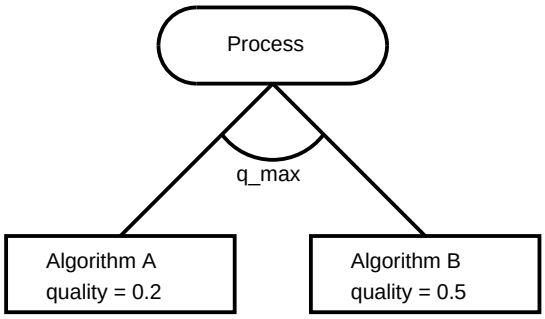

Figure 2: TÆMS representation of the non-boolean OR

could be used in execution when all subtasks must complete for the supertask to produce any quality. It could be used to represent a parallel computation on a segmented set of data where any correct result must use the results of each segment of data. A sum QAF could represent a large scale task where each subtask produces a useful result and is aggregated with others to produce the final result. This could be a graphics scene rendering where the subtasks represent supersampled points of light for a raytrace. The supertask quality is then equal to the number of light points that have completed their raytrace.

These are simple examples, but give us a starting place to examine the benefits of the model. Suppose the raytracing application belongs to a commercial movie studio that is paying for computing power using an on demand market. Also assume that during the movie development phase, the studio requires frequent full-length renderings to gauge progress. The studio has allocated a certain amount of money for the execution and the engine realizes during processing that it will not be able to complete the task on time and under budget. Under current workflow models the engine has no real recourse (other than to perhaps terminate the application or notify someone in charge of the shortcoming). Under a TÆMS model the studio could have told the engine that tracing 90% of the rays is an acceptable quality level for the pre-release renderings (of course 100% is preferred and would be required for the late stages and final release). Given this optimization function and the QAF structure, the execution engine could have chosen to only execute 90% of the subtasks and met the requested quality level. If this 10% reduction in required computing power allowed the application to finish on time and within budget, the engine has accomplished the task that was impossible under current workflow models.

The other TÆMS structure for connections, the node-to-node interrelationship structure, is represented by two types, enables and facilitates (actually four if you include their opposites, disables and hinders which we will not discuss here). The enable relationship specifies an ordering between two tasks. If task A is said to enable task B, then task A must be performed before task B can accumulate any quality. This relationship provides for sequential task ordering essential to current workflow models. The facilitates type specifies a beneficial effect of performing a task. If task A is said to facilitate task B, then by executing task A, task B may have a shorter runtime, lower cost, or increased quality.

The facilitates relationship is not present in current workflow models, but again can improve the reasoning capabilities of the execution engine. For example, an algorithm C can run using any random data set, but produces a higher quality result when algorithm D has preprocessed the data. Using TÆMS, algorithm D facilitates algorithm C, and the workflow engine could decide to execute algorithm D if the increased quality was worth the cost. In current workflow models the boolean evaluation of success means that the workflow engine cannot distinguish between the results of executing D then C or just executing C.

Conclusion

We have explored the idea of a flexible workflow model using the notion of quality of service within grid applications. Through the use of several examples it has been shown that a flexible model offers several advantages over the rigid models present in current workflow languages. In particular, the task relationships in the TÆMS language have been applied to these examples to show that these advantages translate into improvements for grid workflow execution engines. The next logical step would be to extend a currently accepted grid workflow standard with TÆMS structures, or to propose a new model that reflects a combination of both. This model could then be integrated with an advanced workflow execution module and experimental results could be gathered for performance measurements in an actual grid application. This future work would experimentally validate the ideas presented here, and provide a richer understanding of the advantages of flexible grid workflows.

References

Amin, K., and von Laszewski, G. 2003. *GridAnt: A Grid Workflow System*. Website: http://www-unix.globus.org/cog/projects/gridant/.

Bivens, H. P. 2001. Grid workflow. Grid Computing Environments Working Group, Category: Experimental.

Condor Team, University of Wisconsin-Madison, Website: http://www.cs.wisc.edu/condor/manual/. 2005. *Condor Version 6.6.9 Manual*.

Horling, B.; Lesser, V.; Vincent, R.; Wagner, T.; Raja, A.; Zhang, S.; Decker, K.; and Garvey, A. 1998. The taems white paper. Available from http://dis.cs.umass.edu/research/taems/white/.

Krishnan, S.; Wagstrom, P.; and von Laszewski, G. 2002. Gsfl : A workflow framework for grid services. In Preprint ANL/MCS-P980-0802.

OASIS. 2003. Business process execution language for web services. Available from http://www.ibm.com/developerworks/library/specification/ws-bpel/.

UNICORE. 2004. The unicore grid middleware. Available from the UNICORE Forum website: http://www.unicore.org/.